XpkMaster

COLLABORATORS			
	TITLE : XpkMaster		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 28, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

XpkMaster

1.1 Welcome to the XPK distribution

```
  About       About this distribution.
  Philosophy      Some background information.
  SubLibs     Documentation of the sublibraries.
  Contacts      Addresses of people related to XPK.
  C-Utils     Documentation of the included programs.
  XFH       Documentation for XFH system.
  Version Info      What changes are made.
  GNU - License     License for some of added libs/programs.
  Xpk programs      Xpk supporting/using programs.

The contents of the distribution:

xpk_User.lha
  Arexx/      Arexx macros for use with XFH.
  C/        Various programs using XPK.
  catalogs/     Catalog Files for use with locale.library.
  Devs/       Device-information for XFH.
  L/        The XFH-Handler.
  Libs/xpkmaster.library  The heart of the XPK system.
  Libs/compressors/   Compression sublibraries.
  Libs_68020+/      68020+ versions of some sublibraries.
  S/        Settings for XFH.
  XpkMaster.guide   This documentation.

xpk_Crypt.lha (not distributed in USA)
  libs/compressors/   Encryption sublibraries.
  source/     Source files of encryption libs.

xpk_Develop.lha
  Autodocs/     Programmer documentations of libraries.
  Include/      Includes for programmers.

xpk_Source.lha
        Sources to libraries and utility programs.
```

1.2 xpk programs

```
XpkArchive System:
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:  Aminet directory util/arc, filename XpkArchive.lha
```

1.3 about

```
This distribution was created by Dirk Stöcker in October 1996.

I made it up of the newest data I could get, but I think some parts aren't
up to date. Please tell me what's 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 isn't 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
will 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 shouldn't, please tell me and I'll
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 documentation or the catalog file
for xpkmaster.library, send it, and I'll include the stuff in next
release.
```

1.4 history

```
Dirk Stöcker's changes:

3.10: I reworked the source and added locale.library support, an better
library header and an other way of debugging.
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 can't 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 shouldn't 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 doesn't
work. :-( --- I'll have to rewrite it all.

3.3: Files shorten than 4 bytes were not planed for at all. It won't
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 wasn't sufficiently better
than V1.x to warrant a new release.]
```

1.5 XPK - A STANDARD FOR DATA COMPRESSION

```
MOTIVATION
----------
* Many programs that should offer data compression (e.g. HD backup
  utilities) don't.
* Many programs that offer data compression use old, slow, inefficient or
  inappropriate algorithm.
* All programs that offer data compression offer just one algorithm, you're
  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're 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. Therefor an new library interface will be created, the xex libraries.

THE XPK FILE FORMAT

Offers checksums, chunks (important when Seek()s [not yet implemented] 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]
* 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 don’t 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.

1.6 gnu-license

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software
patents. We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary. To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and
modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License. The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language. (Hereinafter, translation is included without limitation in
the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope. The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices
stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in
whole or in part contains or is derived from the Program or any
part thereof, to be licensed as a whole at no charge to all third
parties under the terms of this License.

c) If the modified program normally reads commands interactively
when run, you must cause it, when started running for such
interactive use in the most ordinary way, to print or display an

announcement including an appropriate copyright notice and a
notice that there is no warranty (or else, saying that you provide
a warranty) and that users may redistribute the program under
these conditions, and telling the user how to view a copy of this
License. (Exception: if the Program itself is interactive but
does not normally print such an announcement, your work based on
the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works. But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable
source code, which must be distributed under the terms of Sections
1 and 2 above on a medium customarily used for software interchange;
or,

b) Accompany it with a written offer, valid for at least three
years, to give any third party, for a charge no more than your
cost of physically performing source distribution, a complete
machine-readable copy of the corresponding source code, to be
distributed under the terms of Sections 1 and 2 above on a medium
customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer
to distribute corresponding source code. (This alternative is
allowed only for noncommercial distribution and only if you
received the program in object code or executable form with such
an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it. For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable. However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary

form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is

implemented by public license practices. Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded. In such case, this License incorporates
the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time. Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number. If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation. If the Program does not specify a version number of
this License, you may choose any version ever published by the Free
Software Foundation.

10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission. For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this. Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY

YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989

```
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs. If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library. If this is what you want to do, use the GNU Library General
Public License instead of this License.
```

1.7 xfh

```
      XFH-Handler 1.39

  Copyright (C) 1991, 1992, 1993, 1994 Kristian Nielsen.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License.

Comments, suggestions and bug reports are welcome.

Legal Issues
------------

This program as a whole is distributed under the GNU General Public
License. However, some of the contained material carries different
legal status. In particular, this product includes software developed
by the University of California, Berkeley and its contributors.
Details appear in each individual file in the source directories.

The "XFH" commodity was written by Nicola Salmoria.

This program (the commodity with icons and mountlists) is freely
distributable as long as the archive remains intact, and only a
nominal fee is charged for its distribution. However, it is still
provided "AS IS" without warranty of any kind, either expressed or
implied. By using it, you agree to accept the entire risk as to the
quality and performance of the program.

Version after 1.32 and xScan were done by Matthias Scheler.

Changes since the previous versions
-----------------------------------

Changes since last release V1.34:
 - more 2.0 packets including support for hard- and softlinks
 - extended builtin support for xScan
 - XFH is now distributed with full source code for SAS/C Release 6.51.
 - kludge to get XFH to work on top of Envoy FS

Changes since last release V1.32:
 - After using the supplied tool xScan XFH will read directories MUCH
   faster.
 - XFH now supports ACTION_PARENT_FH. Now MultiView and
   "amigaguide.datatype" have no more problems to use links between
   different AmigaGuide files.
 - XFH is now distributed with full source code for SAS/C Release 6.
```

```
Important changes include since V1.12:
 - XFH is now distributed with full source code (for GCC).
 - The dreaded 'ExNext() / ExAll() bug with filenotes has been fixed.
 - Much improved user interface: Gui 'front panel' (by Nicola Salmoria),
   improved mountlist control.
 - Arexx interface for setting options after mounting the handler.
 - Support for MODE_READWRITE (appending to files).

Overview
--------

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
don't have to care about which compressor was used to create the files.

This version of XFH is designed to work with Xpk, the data compression
standard. You will need Xpk to use XFH. Most users should receive XFH as
part of the Xpk distribution.

[Notes for users of previous versions of XFH: The way the handler is
mounted has changed somewhat since v1.12 in order to make it simpler to
use; you cannot just replace the binary in L: and go. This doc file does
not mention the old option files, but they can still be used if you want
to. Refer to the file advanced_usage.doc. Also note that XFH now supports
updating of existing files, something that a lot of users have requested.]

Installation
------------

This doc file is intended for Workbench 2.0 or later. If you are using
Wb 1.x, refer to advanced_usage.doc. That file contains also information
which may be useful to the advanced user.

We have tried to make the installation of XFH as easy as possible, but it
still isn't a trivial task. Please read this paragraph carefully.

First of all, you will need to have xpkmaster.library installed
(xpkmaster.library is distributed as part of Xpk). Refer to xpkmaster
documentation for installation. Version 2.2 (or any later version) of
xpkmaster.library is suggested, since previous versions contain some bugs
which may make XFH behave incorrectly.

Then, copy "XFH-Handler" to your L: directory, and drag "XFH" into the
Wbstartup drawer. XFH is a commodity, so as usual you will want to change
the CX_POPUP ToolType from YES to NO to avoid having the window open
everytime you boot.

Now you have to decide which partition you want to install XFH upon. Of
course you can choose more than one partition, but with this version of XFH
you shouldn't use the boot partition unless you really know what you are
doing.
>> NEVER COMPRESS XFH-HANDLER, XPKMASTER.LIBRARY, MOUNT, OR ANY OTHER FILE
```

NEEDED TO MAKE XFH WORK!!!!!! <<

You are not limited to install on full partitions; you can choose any directory, but usage on a whole partition is probably the more immediate and useful. Here, we will assume installation on a whole partition. Installation over a directory is accomplished in a similar way, or by modifying the mountlist entry as explained in advanced_usage.

If you are using Workbench 2.1 or later, drag the icon "XDH1" from the "Workbench2.1+" drawer to the "Devs/DOSDrivers" drawer in your boot partition. As its name suggests, XDH1 will work on your DH1 partition. To choose a different partition, just rename the icon; for example, "XJH2" will refer to JH2:, and so on (in fact any letter will work, not just 'X'). You can also use assigned names; for example, let's say you have assigned DOCS: to DH2:text/docs, then a copy of the icon named "XDOCS" will create an XFH task using that directory. To create multiple XFH partitions, use the Workbench 'Copy' command to duplicate the "XDH1" icon and rename it appropriately (ie. to "XDH2"). No other changes are needed.

If you are not using Wb 2.1 yet, you will have to append the sample entry "devs/mountlist.xdh1" to your "DEVS:Mountlist" file (appending one copy for each XFH partition you wish to use). The same considerations made before apply: to change the target partition, edit the mountlist and change the XDH1: line appropriately. Then edit the file "S:User-Startup" and add lines like

```
  Mount XDH1:
```

to automatically mount the XFH partitions at boot time.

If you don't like names like XDH1, refer to "advanced_usage.doc" for a way to use a name of your choice (by modifying the mount entry).

When mounted, XFH will display a new icon on your Workbench screen. For example, let's say that your DH1: partition is labelled "DATA"; XFH will call its partition XFH_DATA. After the first installation, you may relabel the volume as usual (XFH will use a file called '.xfhrc' in its root directory to preserve the volume name across reboots).

The first step of installation is completed. Now reset and check that everyting works correctly. Next steps assume that you have rebooted and everything was ok.

By default, XFH doesn't compress files. To do that, invoke the XFH commodity by using the hotkey (default is control alt x) or by double-clicking its icon. You will be shown a list of all mounted XFH partitions. Choose one, and activate the "Compression" checkbox. Now click on "Select Compressor..." button, and you will see a list of available Xpk compressors. Select your favourite compressor and efficiency and click on "OK". Currently, the best choice is probably NUKE, since it features good compression percentage and very fast decompression.

The "Low Memory" checkbox, when activated, tells XFH to reduce memory usage as much as possible, even if that means reducing speed or compression efficiency (this option is not fully implemented in the current version of XFH-Handler).

When you have set up all the partitions, select "Save" from the "Project" menu if you want to make the changes permanent, and click in the close gadget to hide the window ("Save" will store the settings in the .info file of the commodity).

Now installation should be finished! Try to copy something to XDH1:, and try from CLI to 'list' it in DH1: and in XDH1: to see if it has actually been compressed (of course the file in DH1: should be shorter than the file which is seen thru XDH1:).

You can get also more detailed info.

Hint: if you want to compress all the files in your new XFH partition, the faster way is to make a backup and restore of XDH1:.

Limitations

It should be stressed that a given XFH partition binds to a volume, not to a device. This has consequences if XFH is used on a removable media like a floppy disk. For example, trying to use XDF0 to access DF0 will work, but it will use the disk that was in the drive at the time it was mounted and will not recognise a newly inserted disk.

The figures reported by the shell 'Info' command are somewhat strange. The problem is that it isn't really possible to give sensible figures for 'NumBlocks' and 'NumBlocksUsed' (except scanning the entire XFH partition which would be ridicously slow). Currently, their values are the same as those for the underlying file system.

Suggestions

Remember that, in this release of XFH-Handler, the decompressed file has to stay in memory for all the time the file is open. If you are low on memory, do not compress large files.
Do not compress files which stay open for a long time.
If you are using a printer spooler which creates temporary files on the hard disk (like PrintManager by Nicola Salmoria), make sure they are not automatically compressed by XFH (either turn compression off, or use DH1: instead of XDH1:).

You may not want to have both DATA and XFH_DATA displayed as volumes on the Workbench. To avoid that, edit "S:User-Startup" and add the line

```
Assign DATA: DISMOUNT
```

Moreover, since you may have references to DATA: (for example some assigns) you may want to add this line also

```
Assign DATA: XDH1:
```

which will reroute every later access to the XFH partition. Note that to do this trick, the label of DH1: must *NOT* be DH1 or any other name conflicting with a device name. If it is your case, relabel the volume. After the DISMOUNT trick, you will always access the XFH partition from Workbench, but you will still be able to use both DH1: and XDH1: from CLI.

When doing backups, use DH1:, not XDH1:. This way you will use the compressed data, thus requiring less disks. If your backup program provides compression, turn the option off, since it will only slow things down. Remember also to RESTORE to DH1:, or you will end up with a useless partition!

Future Enhancements

The following are a few loose ideas that may sometime be realised in future versions of XFH:

- Support for other file formats. XFH is currently dependent on Xpk for operation; however original aim was (and still is) a general compressor front-end supporting Xpk, Zoo, Lharc, Lha etc.

- Support for custom formats through AREXX. This would make it possible to write simple AREXX scripts that are called by XFH each time a request is made to open a file that XFH does not recognise. The script can then take over if it can handle the file and call the appropriate conversion programs. Thus, one could take for example a standard gif-to-iff converter and write a simple AREXX script that would make DeluxePaint suddenly read GIF pictures

- Setting of options individually for specific directories and/or files (using AmigaDOS pattern matching). This would make it possible to specify that files named '#?.lzh' must not be compressed, or that directory listnings of ':net/uucp/news/' should not report the correct file sizes (for speedup).

- Making the handler multi-treaded (like the ROM file systems) (multi-treadedness means that a large Read()-request won't block a simple CD command).

- Implementing asyncronous I/O for compression and decompression (overlapping CPU time with IO time for large speedups).

Acknowledgments

XFH owes a lot to all the people that have helped me during development with discussions, criticism, suggestions, bug reports etc. (not to mention the steady demands for new versions when I was a bit slow bringing them out...). I am especially indebted to Nicola Salmoria who wrote the nice gui front-end to XFH, wrote most of this doc file and spent a lot of time discussing the user interface of XFH with me. Many of the improvements in user-friendliness since XFH 1.0 should be attributed to Nicola; any remaining inconveniences or bugs are entirely due to me. My thanks should also go to Urban D. Müller for helping me
start the whole concept of the XFH back in the summer of 1991 - without his help the XFH is not likely to have been realised.

XFH has been developed concurrently with my studies at the University of Copenhagen, Department of Computer Science. The institute kindly provides students with access to electronic mail and news; this also has been

essential in the creation of XFH.

Program history

(In the list, an asterix ('*') denotes BETA version that have not been released and should not be used).

V1.00 Initial release.
V1.00a Bug in XObjExamine() fixed (it sometimes got the name of the root dir wrong). Thanks to Matthias Scheler for reporting this bug.
V1.00b XFH: now obtains the values returned by ACTION_INFO and ACTION_DISK_INFO from the underlying file system. This should help problems with 'zero size file system' as experienced with earlier versions of MFR for example. Thanks to Keith H. Brown for pointing my attention to this problem.
V1.10* Beta version implementing option files and automatic compression.
V1.11* Beta with Xpk password support.
V1.12 New XPKPRIORITY option. Also fixes bugs with bad volume names and a msgport that was unnessesarely public; thanks to Nicola Salmoria for telling me about these problem.
V1.20* First beta with GUI and Arexx support.
V1.21* - Bug fix: Write() to a compressed file opened for reading now fails with a return value of -1L (it used to return 0). Thanks to Stefan Boberg for pointing me to this problem.
- Bug fix: Very nasty bug with file notes that caused XFH to crash the system (happened because dos.library does not preserve the fib->fib_Comment field between calls to ExNext()). Thanks to Anders Holmér for taking the bother sending me "snail mail" to let me know of this problem.
V1.22* Enhanced GUI support. XFH now mounts as a handler (instead of as a disk-device based file system). Setting of options in mountlist.
V1.23* Minor bugfixes; some options to help compatibility with various programs.
V1.30* First version with support for MODE_READWRITE. XFH will now retain protection flags, filenotes and file dates when compressing files. ALLOWAPPEND and COMPRESSREADWRITE option. Write() to MODE_OLDFILE files.
V1.31* ACTION_RENAME_DISK; PORTNAME option; minor bug fixes.
V1.32 Full source provided now under GNU GPL. Source now uses RCS. Changed default for option ALLOWAPPEND to ON. Also changed option KILLSTARTUP to ON per default (the enforcer hits in Format etc. were too bad).
V1.33* ACTION_PARENT_FH required for "MultiView" and "amigaguide.datatype" done by Matthias Scheler
V1.34 support for xScan, source now for SAS/C Release 6
done by Matthias Scheler
V1.35* ACTION_EXAMINE_FH required for "gzip". Reduced code size (sprintf() via exec.library), source now for SAS/C Release 6.51. Debugging output is now done via KPrintF() so you can use "sushi" to intercept it. done by Matthias Scheler
V1.36* ACTION_MAKE_LINK and ACTION_READ_LINK for hard- and softlink support done by Matthias Scheler
V1.37* comments for fast directory scan are now created automatically (option XSCAN)
done by Matthias Scheler
V1.38 added work arround for Envoy's buggy ACTION_FH_FROM_LOCK

```
  (option ENVOYKLUDGE)
  done by Matthias Scheler
V1.39 Fixed bug in builtin XSCAN support (V1.37) which caused XFH to
  crash with error 87000004 sometimes. Thanks to Michael Sülmann for
  the final hint about this bug.
  done by Matthias Scheler
```

1.8 xfh-advanced

What's possible.

XFH, when used with Xpk, makes it possible to store data in compressed format without this being visible to the user or to application programs - XFH will make the compressed data appear like ordinary files. This is by no means a new idea. In MS-DOS world, a (commercial) program called 'Stacker' has been available for some time, which makes it possible to compress a whole partition on a harddisk. On the amiga, people have long used powerpacker along with programs like PPMore to store data files in packed formats (but relying on the application to recognise that the file is in compressed format), and programs like PPPatch have been used to change dos.library to recognise the powerpacker format automatically. However, none of these approaches are perfect in all situations, and so there is room for another alternative - the XFH.

XFH works in conjuction with the Xpk approach to data compression. This in itself gives a number of advantages - a flexible interface to the compressing algorithms, lots of different packers available with the possibility to add new ones etc. But compared to approaches like 'Stacker', there is an additional advantage: access to the compressed data is not limited to XFH - most of the time, the XFH way will be the most convenient way to access data, but if needed, the complete range of Xpk applications is available to the user. The file orientated nature of Xpk also means that XFH - like PPPatch etc., but unlike Stacker - will co-exists nicely with any other Amiga filing system without the need to set up any new partitions or prepare the disk with a special program. In fact, if you happened to have a CD-Rom (read only) stuffed with powerpacked files (or any other format that the current version of Xpk supports), you could dump an XFH unit on top of it and largely forget about the disk being compressed from then on.

So, what is possible is to mount a XFH unit on top of each of one or more standard AmigaDOS directories. It is now possible for programs to access the directories as usual, data being decompressed and optionally compressed as needed in a completely transparent way - programs will never know the difference between compressed and uncompressed files - while still having access to the compressed data using conventional compress/uncompress programs. So the dream is that of doubling the capacity of your hard- or floppydisks for zero cost.

As an example of the possibilities of the XFH, I have been using it for holding various doc files, metafont sources and little used emacs scripts and shell commands on HD, saving in the order of perhaps 60% file space with no noticeable degrading of overall system performance.

What is the catch?

Of course, as everyone knows, nothing comes for free. Obviously, there is a speed penalty to compressing or uncompressing a file. This speed penalty will be highly dependent on the actual compression algorithm and data media used; for example an algebraic compression scheme used with a superfast HD will probably be rather slow, while a fast algorithm may actually result in a speed-up of floppy access since the disk access time saved by the smaller file size more that accounts for the time taken decompressing. And of course with the coming of ever more powerful CPU's the speed will be less of a problem. Some things will always be slow, though. Especially directory listning is a problem, since every single file has to be opened in order to check whether the file is compressed or not. In fact, I'll admit that running XFH from floppy on an unaccelerated amiga will sometimes seem a bit slow. However, XFH was not written to be as fast and small as possible, but rather to be flexible and expandable. And I'm sure a lot of users will find it very useful even on 'small' amigas. On an A3000, XFH runs like a dream, of course. And there is still the possibility for speed improvements in later versions.

Aside from the problem of speed, V1.39 of the XFH comes with a few other limitations that I'm hoping to remove in later versions. Most important is the lack of some of the new 2.0 packets, a lack that will become more severe as more 2.0-only programs start to depend on these packets. Another problem is that XFH in the present version is somewhat memory-hungry, in that it will keep any compressed file completely unpacked in memory as long as that file is open. This is mostly due to limitations in the current Xpk interface, and I'm hoping to remove this in a later release. However, currently this means that it is impossible to open a compressed file if it is larger than available memory.
[One way to solve this would be to have XFH use one of the 'virtual memory' programs that are starting to appear. In fact, I have had one or two reports about this actually working. If someone wants to discuss this I'd very much wellcome it.]

A number of other problems to look out for are detailed in a later section in this doc file. Most of these are things that could be fixed in later versions.

Instructions for use.

XFH is implemented as an AmigaDOS device handler. The 'L:' directory contains examples of other such device handlers, and this is usually also the best place to put XFH (the file 'XFH-Handler', to be precise). Like other handlers, XFH must be mounted before it can be used. This can be done by creating an entry for it in the 'DEVS:Mountlist' file (just the mountlist for short) and issuing the command 'Mount <device>', where <device> is the name given to the device in the mountlist. The supplied file 'Devs/Mountlist.custom' contains example entries for the XFH; it might be convenient to append this file to the end of 'DEVS:Mountlist' (the entries assume an assign 'xfhdir:' to the XFH distribution directory). The most important part of each mountlist entry is the Device name (XH1: etc) and the 'Startup' entry, since these are used to configure the various options of XFH. Further information on the mechanism of mountlists and handlers in general can be found in various places; I will not attempt any lenghty explanation here.

Unlike most other 'normal' device handlers, XFH needs additional

information to function correctly. Most importantly it needs to be told what directory it should use as its root directory (that is, where it should look for compressed files). In the simplest case, this can be done by having an assigning (or device volume name) 'nn:' to this directory and mounting an XFH partition with the name 'Xnn:' (any letter will do, not just 'X'). However, XFH also provides 'options', some of which cannot be controlled by the gui but only through the Mountlist or by using Arexx.

Options can be specified in the mountlist or in an option file, and can be changed dynamically using AREXX. An option is specified as a string following the same conventions as the TOOLTYPES of the Workbench.

If the number of options that should be set are limited, it can be convenient to put them directly in the mountlist entry. This is done by using the 'Startup' keyword. Use a line of the form

```
Startup = "!<opt1>=<value1>!<opt2>=<value2>...!<optN>=<valueN>"
```

(that is, a list of option assignments preceded by '!' (no spaces). The '"'s are optional and the '='s can be replaced with '&'s (this is nessesary to be compatible with some mount commands). See the entry for 'XH0:' in 'Devs/Mountlist.custom' for an example.

For a larger number of options, option files should be used. XFH has the concept of primary and secondary option files. The primary option file is used by putting a line of the form

```
Startup = xfhdir:optionfiles/.xfhrc_1
```

into the mountlist entry (distinguised from the other use of this keyword by the fact that no '!' appears). The filename can be anything you like, of course. The secondary option file is named '.xfhrc' and is placed in the root directory of the XFH unit (uncompressed!). The settings in the secondary option file overrides any settings in the primary option file, but there are some restrictions on the options that can be used in the secondary option file, see below. Both files are optional, the handler will pick default values for any options that are not set by the user.

In the option files, options are given each on a line of its own. Some examples are given in the directory 'Optionfiles'. Be careful when using the secondary option file; this is also used by XFH itself to store information that must be preserved across reboots (notably the volume name for Relabel).

When the handler is mounted, most options can be changed by sending an approproate AREXX command to the handler. The name of the port is the same as the name of the device, though this can be changed (see below). Note that XFH will refuce to mount if it cannot open the AREXX port because of a naming conflict. To set an option, send a command

```
SETOPTION <optstring>
```

to the port, where <optstring> should be in the same format as that used in option files. The directory 'Arexx/' contains example AREXX

scripts that may be usefull. For example the command

```
  rx SetAutocompress XDH1 ON
```

will enable automatic compression on XDH1:.

Boolean options can be specified with "NO" / "OFF" or. "YES" / "ON".
String options are specified by simply putting the string after the
'OPTION=' bit, no quotes are needed.

The available options are detailed below:

```
  ROOTDIR
  VOLUMENAME
  AUTOCOMPRESS
  XSCAN
  PACKMODE
  STEPDOWN
  PASSWORD
  XPKPRIORITY
  TRUNCATEONPACK
  FAILONEXNEXT
  KILLSTARTUP
  COMPRESSREADWRITE
  ALLOWAPPEND
  PORTNAME
  ENVOYKLUDGE
```

Option ROOTDIR:

This option is used to set the name of the directory that XFH: is to
reside in. For example 'ROOTDIR=Work:xfh' would cause XFH: to use that
directory as root. Note that this option can only be used in the primary
option file or directly in the mountlist. When used elsewhere it will
simply be ignored. The default is to use the name of the XFH device itself
with the first character removed (so XDH1: becomes DH1:). [Note that this
is different from XFH v1.12 and earlier.]

Option VOLUMENAME:

This option is used to set the name of the XFH: volume. This is the name
that will be used in absolute path specifications, as well as the one
returned by Info(). Ie. if you go 'VOLUMENAME=Manuals', access can be by
'Manuals:' as well as by 'XHn:' (or whatever). If this option is not
specified, the default is to use the name of the directory in the
underlying file system, or (if this is itself the root of a volume) to
use the name with 'XFH_' prepended. Again, this is mainly for
compatibility with early versions of XFH:. An alternative to using
this option is to use the normal Relabel (from shell or WB); this will
automatically create an entry for this option in the secondary option
file (replacing any existing entry).

Option AUTOCOMPRESS:

This options tells whether XFH: should attempt to compress the files
written through it. When this option is set, everytime a file written to

the XFH: is closed, an attempt will be made to compress the file to a temporary file using Xpk. If this is succesfull, the temporary file will be renamed to the original name and the uncompressed file will be deleted. If the compression fails for any reason, the uncompressed file will simply remain intact. This also means that if anything should go wrong during compression (like a disk full error), it is unlikely that any data will be lost since at least one of the two files should be intact (though possibly with a wierd name). This option is OFF by default.

Option XSCAN:

This option tells whether XFH: should create xScan-like comment automatically. These comments will allow XFH to read directories much faster, see xScan for more information. This option is OFF by default.

Option PACKMODE:

This option selects the mode that Xpk should use when compressing files. It is specified in the usual way when using Xpk. For example, to use 12-bit BLZW compression, 'PACKMODE=BLZW.12' would be used. The default is to use the NUKE compression ('PACKMODE=NUKE'). Of course, to use a specific compression method, the nessesary sublibrary must be available in LIBS:.

Option STEPDOWN:

This option controls the Xpk flag 'XPK_StepDown' during packing. If set, it means that Xpk is allowed to reduce packing efficiency if nessesary to save memory. Refer to the Xpk documentation for details. Default is OFF.

Option PASSWORD:

This option is used to set the password that XFH: should pass on to Xpk when compressing or uncompressing. It should be noted that there is no attempt to keep this password safe from 'memory peekers'. Of course, storing the password in the option file isn't a good idea. A better idea is to use AREXX; the AREXX script 'SetPassword.rexx' in the directory 'Arexx/' may be helpful here. Note that if AUTOCOMPRESS is requested, files may still be saved unencrypted in low-memory or low-diskspace situations, and (depending on the underlying file system / disk device) part of the unencrypted data may still be physically stored on the disk after the deletion of the unencrypted file.

If an attempt to open a file is made when XFH has been given the wrong password, the open will fail with error code 212 (Object wrong type).

Option XPKPRIORITY:

This option is used to set the task priority that should be used when doing Xpk operations (compress/uncompress etc). Setting this to zero or less will prevent XFH from stealing all CPU-time from tasks running at a 'normal' priority. Note that it is possible to set the priority of the handler itself in the mountlist. If this option is not used, the handler will keep whatever priority it is running at when calling the Xpk library.

Option TRUNCATEONPACK:

This option is somewhat technical in nature and can be safely ignored. It is only used in case of an error occuring during the compression of a file. In this case, the compressed file has to be deleted, and if this option is set, the handler will try to call SetFileSize() first to truncate the file to 0 bytes (perhaps saving the flushing of a few buffers). However, due to sparse documentation I'm uncertain whether this feature is stable, and hence it is OFF by default. Again, unless you are curious and don't mind risking crashes/data losses, forget about this option.

Option FAILONEXNEXT:

This option was included to solve a problem with some (or more likely most) programs that perform directory scanning. The problem appears when listing a directory containing a file that is opened with an exclusive lock (for example, MODE_NEWFILE). In this case, XFH cannot determine the correct size of the file and thus fails with an appropriate error code. However, many programs just assume that the end of the directory has been reached. Setting this option to OFF will prevent XFH from reporting failure, returning a fileinfoblock with the wrong file size instead. Since this could result in data loss because programs will se the wrong file sizes, this option is ON by default.

Option KILLSTARTUP:

This option has been included to fix some problems with programs like 'Info' that examine data about file system handlers. It seems that these programs assume that the 'Startup' entry in the device node of any handler that supports volumes will be a FileSystemStartupMsg. However, in the case of XFH, it is a string. This behaviour seems to me to be completely unresonable, and thus a bug in these programs. The only way I could find to fix this was to have XFH manually erase the Startup field in the device node. This is not likely to be an officially supported way of poking the device node. Nevertheless, I had to agree that the problem is intolerable, and thus this option is on by default. However, it can be turned off if you have a mount command that does not like handlers that modify their device node.

Option COMPRESSREADWRITE:

This option controls whether new files opened with MODE_READWRITE will be compressed upon Close(). It is ON by default.

Option ALLOWAPPEND:

This option must be set to allow the writing to existing files (using MODE_READWRITE or MODE_OLDFILE). It is my personal opinion that compressed files shouldn't really be updated in place, since it requires first uncompressing the file on disk, then doing the Write(), then compressing the file again. However, some programs need this ability. This option is ON by default.

Option PORTNAME:

This option can be used to set the name of the AREXX port. This option can only be set in the option file or directly in the mountlist (no, you cannot set it using AREXX...). The default name is the same as the name of the XFH device.

Option ENVOYKLUDGE:

This option tells XFH: not to use ACTION_FH_FROM_LOCk for internal purpose to avoid trouble with the buggy implementation of this packet in Envoy FS. Without this kludge using a XFH: on a Envoy FS volume will crash the FileSystem on the server. The kludge may be not necessary for future version of Envoy. This option is OFF by default.

In case of an error during the scanning of the option files the handler will fail its initialisation and hence refuse to load with error code 114 (Bad Template). Sorry, but there are currently no real error messages implented (this will hopefully be fixed in a later version). If the handler refuses to work for no apparent reason, be sure to tripple-check your option files for errors.

Limitations and known bugs.

The error detection code in the initialisation part of the handler is somewhat flaky - I've tried to make it resonably safe, but documentation on the right way to start a handler is hard to find. What it means is that it is a good idea to make sure that the handler is placed in L: and that the nessesary Xpk libraries are placed in libs: before starting the handler (remember, XFH won't be able to tell you the reason if it was unable to initialise for some reason). Another subtle problem is that due to a quirk of the device list locking, it is vital (using XFH v1.39) that the handler for the directory that XFH is to sit in is already loaded. Usually this will not be a problem; however, if you are using 3rd party filesystem handlers that are mounted after boot-up, you can avoid problems by accessing them before mounting XFH (for example by creating an assign (not late-binding) to them).

It should be noted that the way the XFH makes the same files available by two different routes is not without its problems. One problem is connected to the volume name - XFH tries to be smart about it, but it will sometimes create a duplicate volume name which is a bad idea. To solve this problem the VOLUMENAME option should be used in an option file, or the XFH volume should be changed with Relabel. Another problem when mounting XFH in the root dir appears when using the 'Leave Out' feature of the 2.0 Workbench. Here, the '.backdrop' is duplicated in both volumes, making the left-out icons appear twice. I'm working on a decent solution to this problem. Meanwhile, I would recommend that XFH is mounted only on top of subdirectories.

To provide maximum transparency for application programs, both of the options AUTOCOMPRESS and ALLOWAPPEND should be set. This will make XFH compress files that are written to it 'on the fly', even for programs that update existing files (ie MODE_READWRITE, 'append mode', shell '>>' redirection etc). However, a bit of care is advised when using the ALLOWAPPEND option. For example, it is obviously not a good idea to have two programs writing to the same compressed file at the same time. Even worse, if one program is accessing a file on the underlying file system at the same time that another program is writing to the same file using XFH, XFH is unable to guard completely against data loss. Bevare of this situation. Another thing is that writing to a compressed file opened with MODE_OLDFILE (like Lha does) relies on the ACTION_CHANGE_MODE packet (to partially solve the problems just

mentioned), and thus won't work using KS1.3 file systems or some file systems written before the apperance of KS2.0.

Needless to say (but I'll say it anyway), you should not assign libs: to a XFH unit unless you are absolutely sure of what you are doing. A nice trap is to have the XFH call (and wait for) Xpk, which will then wait for XFH to load a particular library for it.

It should be noted that any given unit of the XFH binds to a directory, not to a DOS device. This means that, currently, it is not possible to have a XFH unit working like DFx: - any access will refer to the disk that was in the drive when the handler was started, not to the disk currently in the drive.

The figures reported by the shell 'Info' command are somewhat strange. The problem is that it isn't really possible to give sensible figures for 'NumBlocks' and 'NumBlocksUsed'. Currently, their values are the same as those for the underlying file system.

XFH has problems with exclusive locks (for example trying to obtain an exclusive lock on "/" or "foo//" will always fail).

Some people have experienced problems when using XFH with the arp.library. This is because a bug/feature in the apr.library function CompareLock() (it declares two locks equal only if the lock->fl_Key fields are equal, which is illegal according to the 2.0 DOS manual). Under 2.0, the program in the directory 'patcharp' can be used to patch arp to use the correct 2.0 SameLock() call. Also, I know of a program that remaps the arp calls to the corresponding 2.0 dos.library calls (though I haven't tried it), it might help too.

Theory of operation.

XFH works by installing itself in the system as a file system handler like DF0: or RAM:. However, unlike most file system handlers, which sit on top of a device (or rest in themselves like RAM:), XFH sits on top of an underlying file system handler (abbrevated to UFS) containing a mixture of normal and compressed files. After initialising itself, XFH sits in a loop receiving packets from AmigaDOS and other applications. Each packet is examined and the appropriate action taken. For example, an open request will cause XFH to open the given file, check whether it is compressed, and if so unpack it to memory for later read requests.

XFH is currently single-threaded, unlike the Commodore file systems (this means that it is not possible for the XFH to service other requests while waiting for the UFS). I'm hoping to fix this in some later version.

Acknowledgments.

The author wishes to thank all the people that have participated in the development of Xpk without which the XFH would not have been the same. I am especially grateful to Urban D. Müller for helping me start the whole concept of the XFH back in the summer of 1991 - without his help the XFH is not likely to have been realised. Thanks also to the many beta testers who helped me iron out as many bugs as possible before release; your help work has been very valuable to me. And thanks must go, of course, to the guys at Commodore for bringing to us the wonderful

Amiga.

1.9 xfh-links

Since version 1.36 XFH supports hard- and softlinks.

XFH does this by passing the required packets to the underlying file system, it does NOT create or handle links itself.

In most cases this underlying file system will be Commodore's file system. Versions before V40 of this file systems have SERIOUS BUGS in handling of hardlinks which might cause CORRUPTION of your volumes.

If you create a hardlink to a file on such a file system and remove this file while the hardlink still exists you will have a corrupted directory. So BE CAREFUL with hardlinks. So: Don't try this at home, kids. ;-)

The implementation of softlinks is not as it should be but you don't risks your data by using softlinks. The only problem is all released versions of Commodore's "MakeLink" command do NOT support creating softlinks. If you want to create softlinks use one of public domain replacements or any other program that is able to create softlinks.

1.10 Documentation of the included sub libraries

```
  BLZW  Bryan's turbo-charged LZW compressor
  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
```

Installation: copy the files to directory LIBS:compressors !

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 blzw

BLZW is an XPK packer sublibrary which implements a highly optimized
form of the popular LZW compression algorithm. This is essentially the
same algorithm used in Arc, Zoo, and Unix compress.

Most common packers for the Amiga are oriented toward pack-once,
unpack-many situations (games, demos, etc.), and thus concentrate on
unpacking speed while sacrificing decompression. Unlike these, BLZW is
intended for situations where packing speed is also important: for example,
in real-time communications (i.e. compressing data going into a modem and
decompressing it as it comes out the other end), or hard drive backup.

The LZW algorithm can operate in several modes, or "maximum code
sizes." Without getting into too many technicalities, larger codes
generally result in better compression ratios (especially for large files)
and slightly higher speed, but require more memory for both packing and
unpacking. BLZW can be told to use any maximum code size from 9 to 15
bits. For comparison, Arc used an LZW algorithm with a maximum code size
of 12 bits, Zoo operated at 13 bits, and Unix compress commonly operates at
14 or 16 bits. (Supporting 16-bit compression in BLZW would have
necessitated significant slowdown, so I decided not to support it for now.)

To select the maximum code size BLZW uses, look at the table below,
find the code size you want (or just look for the mode with the statistics
you would like), and select any number within the listed "mode range". Then
append a period and the mode number to the "BLZW" keyword specified to XPK
(or any other program that lets you specify an XPK method). For example,
to get 14-bit compression, you could type "XPK BLZW.80 <infile> <outfile>".
If you specify no maximum code size, BLZW defaults to 13 bits, and hence
operates at basically the same compression ratio as Zoo (although much
faster).

Following is a table briefly listing some comparative statistics for
BLZW. These were generated by xbench on the standard XPK benchmark system
(A3000/25 with SCRAM, using the AmigaVision executable as data).

Code Size	Mode Range	Packing Memory	Unpacking Memory	Packing Speed	Unpacking Speed	Compression Ratio
9	0-14	3K	2K	159 K/s	303 K/s	24.4%
10	15-28	7K	4K	141 K/s	328 K/s	29.4%
11	29-42	15K	8K	135 K/s	343 K/s	31.7%
12	43-57	30K	16K	134 K/s	356 K/s	32.4%
13	58-71	60K	32K	139 K/s	364 K/s	32.9%
14	72-85	120K	64K	143 K/s	374 K/s	33.1%
15	86-100	240K	128K	157 K/s	381 K/s	33.7%

Version History:

```
3.00  (R3, 24-Sep-92)
  Fixed a bug in the compressor that would generate a bad file if
      the last code written needed to expand the code size.
2.00  Hmmm, dunno exactly what happened to this version...
1.00  (R2, 4-Jun-92)
  First public release.

Contact Address: Bryan Ford
```

1.12 cbr0

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%.

Following is a table briefly listing some comparative statistics for CBR0 These were generated by xBench on the standard XPK benchmark system (A3000/25 with SCRAM, using the AmigaVision executable as data). Note that memory needs don't include xpkmaster.library's buffers.

Method	Packing Memory	Unpacking Memory	Packing Speed	Unpacking Speed	Compression Ratio
RLEN	??? K	??? K	139 K/s	526 K/s	4.1%
CBR0	0 K	0 K	410 K/s	1918 K/s	3.1%

I tried to compare both libraries (CBR & RLEN) against each other on some really long ASCII files (up to 500k), but I don't write those compression results here for I can't know if all of you have those files to check my results. The speed factors given below are representative for most files but compression factor differs on both libraries depending on the files. Sometimes RLEN obtains better results, sometimes CBR0. You better compare them on your own and decide then which one you prefer. But always keep in mind that my CBR0 is over 3 times faster on decompression than RLEN.

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.
```

1.13 dlta

```
History:  v0.1  first release

FEATURES
-supports the XPK standard
-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's 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).

1.14 duke

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

1.15 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.

Following is a table briefly listing some comparative statistics for
most of the xpk compression sublibraries. These are the results of the XPK
standard benchmark xBench on the standard XPK benchmark system A3000/25
with SCRAM, using the AmigaVision executable as data. Note that memory
requirements don't include xpkmaster.library's buffers. You can get at the
results for other libraries with the help of xQuery.

Method	Mode	Packing Memory	Unpacking Memory	Packing Speed	Unpacking Speed	Compression Ratio
FAST	0..79	64K	0K	428 K/s	1055 K/s	32.7%
FAST	80..100	272K	0K	70 K/s	1096 K/s	39.3%
RDCN	0..100	16K	0K	217 K/s	800 K/s	33.2% *
BLZW	0..14	3K	2K	159 K/s	303 K/s	24.4%
BLZW	15..28	7K	4K	141 K/s	328 K/s	29.4%
BLZW	29..42	15K	8K	135 K/s	343 K/s	31.7%
BLZW	43..57	30K	16K	134 K/s	356 K/s	32.4%
BLZW	58..71	60K	32K	139 K/s	364 K/s	32.9%
BLZW	72..85	120K	64K	143 K/s	374 K/s	33.1%
BLZW	86..100	240K	128K	157 K/s	381 K/s	33.7%
NUKE	0..100	192K	0K	35 K/s	613 K/s	45.2%

```
IMPL  0..10   300K        0K       29 K/s   360 K/s 34.8%
IMPL   11..30   350K        0K       27 K/s   332 K/s 39.8%
IMPL   31..50   400K        0K       20 K/s   314 K/s 43.3%
IMPL   51..75   425K        0K       14 K/s   300 K/s 44.0%
IMPL   76..98   450K        0K  8 K/s   292 K/s 44.2%
IMPL   99..100    450K        0K  6 K/s   291 K/s 44.3%

RLEN  0..100      0K        0K      140 K/s  1043 K/s  4.5%
CBR0  0..100      0K        0K      388 K/s  1833 K/s  3.1% (**)
```

(*) The results compiled into xpkRDCN.library are wrong! [The author of RDCN did not have access to a Amiga3000/25 with SCRM and had to guess.] The results presented here have been newly measured and represent the behaviour of the xpkRDCN.library V2.2.

(**)Same as (*) for xpkRDCN.library.

Some Comments to the above table: Always remember that these comments are just an interpretation of the above table. There are probably data files giving totally different results!

* RDCN is FAST's direct competitor, it gives a bit more compression, but is significantly slower.
* If you need a very fast decompression use FAST.
* For symmetric applications use either FAST or BLZW.
  [BLZW is always two to three times slower than FAST, but is better in compressing text files.]
* Do not use IMPL, NUKE is faster and gives better compression.
* Don't expect too much compression from run length compressors like RLEN or CBR0. If you want to use a runlength encoder use CBR0, it's much faster than RLEN.

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

1.16 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.)

Speed and Memoryusage

Rounds	Memory Usage	En-/Decryptioncryption Speed
4	1K	190 K/sec
8	1K	144 K/sec
16	1K	96 K/sec
32	1K	58 K/sec
64	1K	33 K/sec

Safety

Rounds	Safety (? ;-)
4	unsafe, broken (Murphy 1990)
8	unbreakable for ``normal'' people
16 (default)	good Cryptoanalysists can decypher this with less 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 don't 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       |
  +-----------------------------------------------------------+
```

1.17 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

Following is a table briefly listing some comparative statistics for HFMN. These were generated by xBench on the standard XPK benchmark system (A3000/25 with SCRAM, using the AmigaVision executeable as data) and on A4000/40 (Booting without Startup-Sequence, with Setpatch). Note that memory needs don't include xpkmaster.library's buffers.

Method	Packing Memory	Unpacking Memory	Packing Speed	Unpacking Speed	Compression Ratio
HFMN.000+	7.5 K	5 K	223 K/s	209 K/s	24.7
HFMN.020+	7.5 K	5 K	259 K/s	209 K/s	24.7

and now the same with A4000/40

Method	Packing Memory	Unpacking Memory	Packing Speed	Unpacking Speed	Compression Ratio
HFMN.000+	7.5 K	5 K	537 K/s	569 K/s	24.7

```
      HFMN.020+    7.5 K      5 K  592 K/s    569 K/s    24.7
```

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's 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 couldn't 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
        wasn't handled correctly)
       Thanks to David Balazic for reporting this one.
    V 1.36 - 1.35 bugfix wasn't 100% ok.
```

Contact Address

Martin Hauner

1.18 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 won't 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 it's 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 it's internal huffman tree. Therefore it doesn't 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 won't, 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'm 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.

Statistics

Following is a table briefly listing some comparative statistics for HUFF without encryption. These were generated by xBench on the standard XPK benchmark system (A3000/25 with SCRAM, using the AmigaVision executable as data). Note that memory needs don't include xpkmaster.library's buffers.

Method	Packing Memory	Unpacking Memory	Packing Speed	Unpacking Speed	Compression Ratio
HUFF	30K	71K	88 K/s	138 K/s	24.1%

Where unpack speed varies depending on decrunch code (refer to source for that).

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 don't know any bugs in this version, but if you should meet one, please let me know via email (refer to end of this document for my email adr). 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 don't have modes
;       yet (but I have a mapping code :-=))
```

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

  Marc Zimmermann
```

1.19 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's 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 it's 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 it's 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's sure that it's 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's more complicated to crack messages crypted with this code, because one MUST start at the beginning of the text. It's 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:

XPK Mode  Encr. Method  Nr. States  68030/25  68000/7.14
--------  ------------  ----------  --------  ----------
 0..25    ECB   /   90 K/s    12 K/s
-------------------------------------------------------------------------
 26   CFB   1
  .    .    .   87 K/s    11 K/s
  .    .    .
 50   CFB   25
-------------------------------------------------------------------------
 51   OFB   1
  .    .    .   84 K/s    11 K/s
  .    .    .
 75   OFB   25
-------------------------------------------------------------------------
 76   CBC   1
  .    .    .   84 K/s    11 K/s
  .    .    .
 100    CBC   25
-------------------------------------------------------------------------

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 don't 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's 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.

Version History:

1.00  ( 5-Aug-92 )  First public release.

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
```

1.20 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).

Following is a table, listing some comparative statistics for executables, for all compression modes, using a xpk chunk size of 64K. These were generated by xBench on the standard XPK benchmark system (A3000/25 with SCRAM, using the AmigaVision executable as data). Note that memory requirements do not include xpkmaster.library's buffers. The 2nd number indicates the memory needed using the non-turbo mode, which automatically kicks in if there's insufficient memory available. Thanks to Urban Dominik Müller for providing this information.

Method	Mode	Packing Memory	Unpacking Memory	Packing Speed	Unpacking Speed	Compression Ratio	Description
IMPL	0..10	300K/0K	0K	29 K/s	360 K/s	34.8%	0.10*max
IMPL	11..30	350K/0K	0K	27 K/s	332 K/s	39.8%	0.30*max
IMPL	31..50	400K/0K	0K	20 K/s	314 K/s	43.3%	0.50*max
IMPL	51..75	425K/0K	0K	14 K/s	300 K/s	44.0%	0.75*max
IMPL	76..98	450K/0K	0K	8 K/s	292 K/s	44.2%	1.00*max
IMPL	99..100	450K/0K	0K	6 K/s	291 K/s	44.3%	adaptive

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.

What does this 1.00*max description mean? First the maximum allowable compression mode, for a particular chunksize, is computed. Then this maximum mode is scaled down by a factor depending on the chosen compression mode (<99). The above table is valid only when using the default chunksize. If a program uses a smaller chunksize the compression speeds will go up, if it uses a larger chunksize compression ratios will improve somewhat.

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's 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

1.21 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'm 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 don't have to worry, this library will also
decrunch old format. Send me an e-mail what you'd 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 :-)

Here is a small benchmark list for those bechmarks' lovers :-)

Evaluated on a A3000/30/25 with 2MB ChipMem and 4MB SCRAM [standard
XPK benchmark system] by XBench using AmigaVision [594712 bytes]

Packer 68020	UComp	Comp	CF	CTime	CSpd	UTime	USpd
mash.100	594712	313612	47.3%	27.86	21346	1.42	418811
mash.30	594712	322012	45.9%	7.10	83762	1.45	410146
mash.0	594712	332360	44.2%	5.58	106579	1.49	399135

Packer 68000	UComp	Comp	CF	CTime	CSpd	UTime	USpd
mash.100	594712	313612	47.3%	27.96	21270	1.47	404565
mash.30	594712	322012	45.9%	7.18	82828	1.50	396474
mash.0	594712	332360	44.2%	5.65	105258	1.54	386176

I hope you like this :-)

Slower mode of packing has the speed of version 1.26
(it's activated when is not possible to allocate 256KB of RAM for large buffer)

	UComp	Comp	CF	Time	KB/s
mash.100	594712	313908	47.3%	65.42	9090
mash.000	594712	332652	44.1%	8.59	69233

"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's 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'm releasing this
  version.
1.98  Well many new checks have been added to prevent a to deep scan
  when better match can't be found. Even a little bit better
  alghorithm was used for lazy_eval -> better CF.

Contact Address: Zdenek Kabelac
```

1.22 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.
```

1.23 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'd say I've come close to the best that can be
done with this approach. There's a drawback, though. The compression must
use large hashing tables to reach this speed. I've made sure that NUKE is
still usable on a plain 512K Amiga, but you won't be able to run many
things besides NUKE while you're packing. There is, by the way, no increase
in mem needs with increasing file size.

Following is a table briefly listing some comparative statistics for NUKE.
These were generated by xBench on the standard XPK benchmark system
(A3000/25 with SCRAM, using the AmigaVision executable as data). Note that
memory needs don't include xpkmaster.library's buffers.

Method   Packing   Unpacking   Packing   Unpacking   Compression
   Memory     Memory  Speed    Speed  Ratio
------   -------   ---------   -------   ---------   -----------
```

```
 NUKE    192K        0K      35 K/s   613 K/s 45.2%


Version History:
1.0 First public release.
1.2 Does compress slower, but a bit better.
  Decompression is faster than V1.00.
1.3 Fixed excessively long 2 byte matches [there were files, on which
  NUKE was not bijective!]

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

1.24 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

        Statistics
        ----------

Following is atable brieflylisting some statistics for RAKE. They were
generated by xBench on an A3000/25 with 2+8 MB Mem, using the AmigaVision
executable as data (standard xpk benchmark configuration). Note that
memory needs don't include xpkmaster.library's buffers.

Filesize uncompressed: 594712 bytes

68020 version:              Mem usage [Kb]
Packer   Comp CF    CTime  CSpd UTime  USpd Pack/Unpacking
-------------------------------------------------------------
rake.100 322872 45.8% 6.64  89565 0.90 660791 256   0
rake.075 324964 45.4% 5.73 103789 0.91 653529 256   0
rake.050 326908 45.1% 5.13 115928 0.91 653529 256   0
rake.025 328796 44.8% 4.77 124677 0.92 646426 256   0

68000 version:              Mem usage [Kb]
Packer   Comp CF    CTime  CSpd UTime  USpd Pack/Unpacking
-------------------------------------------------------------
rake.100 322872 45.8% 7.07  84117 1.16 512682 256   0
rake.075 324964 45.4% 6.16  96544 1.17 508300 256   0
rake.050 326908 45.1% 5.56 106962 1.17 508300 256   0
rake.025 328796 44.8% 5.20 114367 1.18 503993 256   0

        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
```

1.25 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.

Here are some benchmarks to give you an impression of the compressionrate compared to other compression-libraries (BLZW,NUKE,RAKE 68020 V1.5) These tests were done on an A4000/040.

Binary (assembler of SAS-C6.51):

Packer	UComp	Comp	CF	CTime	CSpd	UTime	USpd
blzw	104540	67144	35.8%	0.63	163659	0.16	634266
nuke	104540	53672	48.7%	2.15	48562	0.08	1244520
rake	104540	52980	49.4%	0.75	138040	0.07	1341530
shri.0	104540	48404	53.7%	2.94	35484	1.57	66409
shri.15	104540	47544	54.6%	2.97	35094	1.53	68291
shri.29	104540	46652	55.4%	3.13	33343	1.50	69491
shri.43	104540	46236	55.8%	3.40	30724	1.43	72610
shri.57	104540	46080	56.0%	3.69	28311	1.42	73229
shri.71	104540	45952	56.1%	4.11	25425	1.40	74198
shri.85	104540	45732	56.3%	4.71	22182	1.40	74430

English text:

Packer	UComp	Comp	CF	CTime	CSpd	UTime	USpd

```
blzw     284397 140160 50.8%      1.39  204168      0.42  662889

nuke     284397 139540 51.0%     11.21   25349      0.26 1082039

rake     284397 131760 53.7%      3.85   73740      0.23 1195470

shri.0     284397 137008 51.9%      8.68   32757      4.72   60228
shri.15    284397 131576 53.8%      8.30   34254      4.23   67146
shri.29    284397 126456 55.6%      8.17   34799      3.76   75561
shri.43    284397 121336 57.4%      8.52   33349      3.32   85499
shri.57    284397 116380 59.1%      9.61   29578      2.96   95968
shri.71    284397 112132 60.6%     11.90   23896      2.69  105528
shri.85    284397 108028 62.1%     16.84   16886      2.52  112634
```

C-sourcecode:

```
Packer      UComp   Comp    CF     CTime    CSpd     UTime    USpd
-------------------------------------------------------------------
blzw     102137  38400 62.5%      0.44  230525      0.14  709441

nuke     102137  28300 72.3%      1.66   61493      0.06 1655974

rake     102137  24916 75.7%      1.30   78178      0.05 1889131

shri.0     102137  30088 70.6%      2.01   50606      0.94  107935
shri.15    102137  27228 73.4%      2.02   50511      0.83  122441
shri.29    102137  25460 75.1%      2.19   46441      0.74  136683
shri.43    102137  23984 76.6%      2.59   39410      0.68  149844
shri.57    102137  22528 78.0%      3.31   30803      0.60  168234
shri.71    102137  21360 79.1%      4.66   21906      0.55  183074
shri.85    102137  20704 79.8%      5.94   17173      0.53  189540
```

As you can see, 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

1.26 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.

So, how well it compresses samples? I took 1.7 MB of samples and ran them thru several packers. The compression ratios I got:

```
  HUFF  17%   DLTA+HUFF 27%
  IMPL  21%   DLTA+IMPL 23%
  NUKE  20%   DLTA+NUKE 23%
  SHRI  29%   DLTA+SHRI 34%
  SMPL  30%   DLTA+SMPL 25%
```

From above table you should see why I chose huffman for compression. It gains most from delta encoding. But if you surely want best compression ratios regardless of time used then go for DLTA+SHRI.

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

Following is a table briefly listing some comparative statistics for SMPL. These were generated by xBench on the standard XPK benchmark system (A3000/25 with SCRAM, using the AmigaVision executable as data). Note that memory needs don't include xpkmaster.library's buffers.

Method	Packing Memory	Unpacking Memory	Packing Speed	Unpacking Speed	Compression Ratio
SMPL	14K	7K	151 K/s	354 K/s	6.7%

Version History: 1.00 First public release.

Contact Address: Jorma Oksanen

1.27 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's 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's 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 convinced me to program the 68000 version.

Statistics

As I don't own a A3000 and xBench, I cannot give results which you can compare with the other XPK libraries. But I can offer you some results I got on my own machine (that is a A2000 with a 22Mhz 68030 GVP with 6 Mb of 60ns 32bit ram)

Frequency	Original	SQSH = Squash	LhA
Sample 15 KHz	732984	418144 (43.0%)	510377 (30.4%)
Sample 20 KHz	979990	532512 (45.7%)	664365 (32.2%)
Sample 25 KHz	1219832	634488 (48.0%)	805746 (33.9%)
Sample 30 KHz	1465864	728656 (50.3%)	940955 (35.8%)
Sample 35 KHz	1710140	815652 (52.3%)	1066660 (37.6%)
Sample 40 KHz	1959924	898416 (54.2%)	1186449 (39.5%)
Sample 45 KHz	2208002	976340 (55.8%)	1294084 (41.4%)
Sample 50 KHz	2456810	1048384 (57.3%)	1389138 (43.5%)
Sample 55 KHz	2725502	1144420 (58.0%)	1506860 (44.7%)

	** SQUASH **			** LHA **		
FileName	Original (in bytes)	%	Pack K/s	Unpack K/s	%	Pack K/s	Unpack K/s
Intuition (AutoDoc)	283460	60.9%	18.7	369.1	66.7%	31.9	247.2

```
MUI.guide         80052  |  58.7%  22.9  312.7  |  65.0%  26.3  211.3
AmigaVision      594712  |  42.4%  22.1  276.6  |  48.7%  18.4  206.7
VirusChecker      42380  |  30.2%  26.0  197.1  |  38.0%  20.8  153.3
Mod.February symph   240420  |  27.8%  29.7  335.4  |  24.2%  25.0  166.5
Mod.Infinite finit   180950  |  41.6%  28.5  353.4  |  38.7%  20.7  182.2
Mod.Began in Africa  197038  |  41.5%  31.2  356.3  |  37.1%  21.6  194.3
Mod.Poing         94530  |  35.1%  15.7  329.7  |  33.8%  27.2  171.0
Mod.Resonance 2      188354  |  43.5%  29.2  375.4  |  35.9%  21.9  182.1
Mod.Tranze seven     215872  |  43.5%  28.5  351.4  |  36.3%  22.4  189.1
OnlyWithYou.8SVX    1219832  |  48.0%  30.2  385.5  |  33.9%  29.4  180.2
-------------------------------------------------------------------------

Contact Address: John Hendrikx
```

1.28 c-utils

```
  xDir
  xDrop
  xLoadSeg
  xPack
  xPK
  xQuery
  xType
  xUP
  xScan
```

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.

HISTORY
  xDir 1.0
  - First public release

  xDir 1.1
  - Enforcer hit fixed
  - Version string added
  - This doc added

COPYRIGHT
  Freely distributable for noncommercial use.
```

```
AUTHOR
  Urban Dominik Müller
```

1.30 xdrop

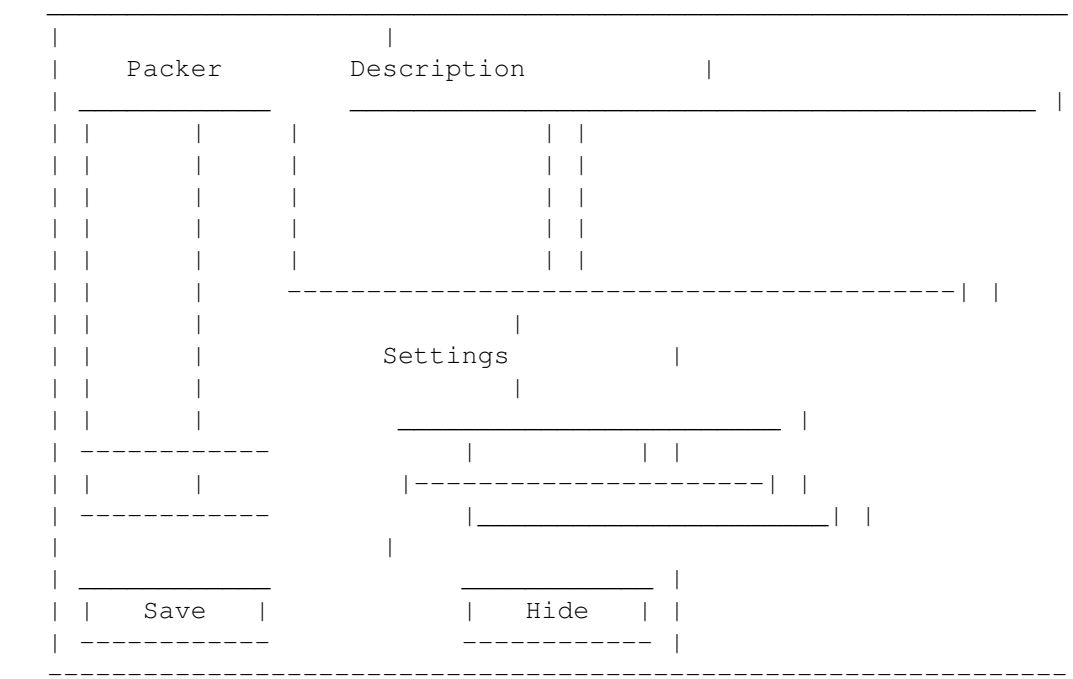
```
XPK-based Workbench packer/unpacker.

Versions
2.0  - First version, distrbuted with xpk 1.0
2.01 - Small bug fix (original window positioning was wrong)
2.02 - Memory leek fixed
2.1  - Layout slightly changed, larger listview, mode descriptor
2.11 - Bug causing Enforcer hit in panel disable code fixed
       (Ch. Schneider)
2.12 - Bug with long filenames fixed
2.20 - New option 'Keep file dates'
2.21 - Now also keeps protection bits and file comments, displays
       name of selected packer if no icon name is specified in
       tool types

Disclaimer

xDrop ist Copyright © 1992 by Martin A. Blatter. All rights reserved.

xDrop may be freely distributed for non-commercial purposes only!

The entire risk as to the quality and performance of this program is
with you. The author assumes no responsibility or liability whatsoever
with respect to your use or inability to use of this software.

Purpose

xDrop is an easy to use visual user interface to the XPK library
system. It allows you to choose from different packers and
different operation modes by a simple mouse click.

It sports a Style Guide compliant user interface and uses Workbench
2.0's appicon and appwindow features to make operation as simple and
convenient as possible. Through the use of Commodore-Amiga's
commodities.library, xDrop can be installed on any hotkey and
fully controlled with the Commodoties Exchange program.

Requirements

· Any Amiga
· Kickstart 2.0 or higher
· The xpk package (xpkmaster.library and at least one sublibrary)

Start xDrop

xDrop can be used either from Workbench or from the CLI.

* Workbench use
```

```
  Double click on the xDrop icon. An appicon will appear on the
  Workbench screen. If this appicon doesn't appear several things could
  have gone wrong:

  · Do you use Kickstart 2.0 or higher?
  · Is there any memory left on your system?
  · Do you have commodities.library (supplied with Workbench 2.04)
    in libs:?
  · Is the xpk package properly installed?

  Requesters will try to tell you the cause if xDrop fails to run.

  If the tool type 'CX_POPUP' (see below for an explanation of tool
  types) is set to 'yes', a configuration window will appear. See
  the section entitled 'Configuration Window' for more information
  about this window.

* CLI use

  Type 'xDrop' in any Shell window. An appicon will appear on the
  Workbench screen.

  To remove xDrop from memory, just press CONTROL-C in the Shell
  window where xDrop has been started from.

Operation

Operation of xDrop is very easy:

To compress files:

   Just select one or more file icons (no drawers) on the Workbench,
   drag them to the xDrop appicon or the xDrop configuration window
   (if it's open) and drop them.

   The Progress Report Window (see below) will appear and the file
   will be compressed using the packer selected in the configuration
   window or pre-set with the XPK_METHOD tool type (see below for
   a discussion of tool types).

To uncompress files:

   To uncompress a compressed file, drag its icon on the xDrop
   appicon or the xDrop configuration window (if it's open).

   You may drop arbitrary many icons, compressed or uncompressd
   simultaneously on xDrop. Note that xDrop currently only handles
   single files (no drawers). You cannot concatenate several files
   to one archives like with lharc or zoo.

To remove xDrop from memory:

   xDrop can be removed from memory at any time.

   - Use the 'Commodities Exchange' program to kill xDrop
   - Send xDrop a CTRL-C signal with the 'break' command.
     xDrop will exit gracefully.
```

- Run another copy of xDrop. Both xDrop processes will quit immediately.

Configuration Window

The configuration window is divided into three visually separated parts labelled ‘Packer’, ‘Settings’ and ‘Description’:



1. Packer

The Packer list view allows you to choose from a list of available XPK packers and encryptors by clicking on its entry. (The packer libraries are usually located in the drawer ‘libs:compressors’). The recessed box below the list view shows the selected packer

2. Description

This view-only box displays more information about the currently selected packer/encryptor consisting of the full name of the packer and two lines of additional information.

3. Settings

The Settings part of the configuration window consists of two gadgets that can be ghosted, depending of the capabilities of the currently selected packer:

A. Efficiency

This proportional gadget allows you to manipulate the efficiency of the packer. The value for efficiency can be in the range from 0 to 100%. Neither must every packer support 100 different levels nor does every packer have more than one level.

An ASCII name of the current efficiency level can be obtained
by selecting the 'Mode Info' menu.

As of version 2.1, the ASCII name is now also being displayed
at the left side of the proportional gadget

B. Password

This string gadget allows you to specify a password for
packers that support data encryption.

The password may not be longer than 15 characters.

4. Buttons

A. Save

Saves the current settings to the xdrop disk icon.
Keyboard equivalent: S

B. Hide

Hides the configuration window but doesn't quit xDrop.
Uses the current configuration settings but doesn't save
them.
Keyboard equivalent: H

5. Close gadget

The window close gadget does exactly the same as the 'Hide' button.

6. Menus

A. Project

About - Displays information about the program and its author
  Keyboard equivalent: Rt. Amiga - A

Mode Info - Display settings and additional information for
the currently selected packer.
Keyboard equivalent: Rt. Amiga - I

Hide - Hides the configuration window but doesn't quit xDrop.
 Uses the current configuration settings but doesn't save
 them.
 Keyboard equivalent: Rt. Amiga - H

Quit - Completely removes xDrop from memory.

B. Option

Keep Original - If checked, xDrop will save the original
    file and append '.xpk' to the filename
    for the compressed version of the file.
    Icons will be copied as well. The '.xpk'
    extension will be deleted when the file
    is going to be decompressed.

```
          If the menu item is not checked, xDrop
          will overwrite the original with the
          compressed version.

      Keep File Date - If this option is enabled, the compressed
           file will get the same date stamp as the
           original.
           This is very useful if you're processing
           files (e.g. different versions of a source code)
           and you're depending on the file date to tell
           which version is newer.

Progress Report Window

While packing/unpacking, xDrop will show a progress report in a special
window that appears in the upper left corner of your Workbench screen.
A bar that moves from the left to the right shows the degree of completion.
This  will not work with all packers, some will only show a change when
done. The progress report also shows the amount of bytes already processed,
the total amount of bytes to be processed and the (de)compression speed.

(De)compression can be aborted at any time using the close gadget;
you will not lose any data. xDrop never overwrites the original before
having sucessfully completed the (de)compression.

The following tool types are supported:

CX_POPUP
  If you want the xDrop configuration window to pop up the
  first time you double-click on the xDrop disk icon, set
  this to yes:
    CX_POPUP=yes

  If you want xDrop to run in the background the first time it is
  being started e.g. if you're running it from the WBStartup drawer,
  set this to no:
    CX_POPUP=no

CX_POPKEY
  Key combination for the commodity 'hotkey'. Default:
    CX_POPKEY=alt shift f9

CX_PRIORITY
  Commodity priority. Default value:
    CX_PRIORITY=0

XPK_METHOD
  Sets the default packer. This tool type will be maintained by the
  configuration window as should not be changed by the user.
    XPK_METHOD=NUKE

XPK_PRIORITY
  Sets the task priority of the packing/unpacking task. Don't change
  this value unless you know what you're doing.
    Default value: 0
```

```
LANGUAGE
  Determines the language to use for the user interface
  Possible values:
    LANGUAGE=German
    LANGUAGE=English
  This tool type overrides global language settings. Remove
  it if you want to use the system's default language...

ICONNAME
  Allows you to specify the label string of the appicon.
  Example:
    ICONNAME=Slurp

  If this Tool Type is not given, the appicon will get the
  name of the currently selected packer.

ICONXPOS
  Sepcifies the desired x coordinate of the appicon. Note that
  as of Kickstart 2.0, Workbench may move your icon to another
  position than specified here.
  Example:
    ICONXPOS=600

ICONYPOS
  Sepcifies the desired y coordinate where the appicon should
  appear. Note that as of Kickstart 2.0, Workbench may move your
  icon to another position than specified here.
    ICONYPOS=400

KEEPORIGINAL
  If this tool type is set to 'yes', xDrop will save the original
  file and append '.xpk' to the filename for the compressed copy of
  the file. Icons will be copied as well. The '.xpk' extension will
  be deleted when the file is going to be decompressed.

  If the tool type is set to 'no' (the default), xDrop will
  overwrite the original with the compressed version.
  Example:
    KEEPORIGINAL=no

  If this tool type is not specified, it defaults to 'no'.


KEEPFILEINFO
  If this tool type is set to 'yes', the processed file will get the
  same date stamp, file comment and protection bits as the original.
  Otherwise, new files will get the current date and time and default
  protections.
  Example:
    KEEPFILEINFO=yes

  Default value is 'no'.

Credits: Author Martin A. Blatter

Bug reports or suggestions are welcome but *please* use e-mail or snail
mail (no phone calls!). Thanks.
```

This program uses Relog AG's ITools(tm), the object-oriented user
interface system by Christian A. Weber.

Special thanks to U. Dominik Müller for parts of the manual.

1.31 xloadseg

```
        WHAT IT DOES
        ------------
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 'XpkLoadSeg' from your startup-sequence. No need to 'run' it.
If you want to remove it, use 'XpkLoadSeg -q'. The 700 bytes will be lost.
Do not try to start from Workbench.

          HISTORY
          -------
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.
1.0 - First release, based on PPLoadSeg by Nico François

            LEGAL MUMBO-JUMBO
            -----------------
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!

        PLEASE NOTE
        -----------
Never pack crucial data (sources, texts) using XPK.  And you better leave
often used files unpacked for speed and safety.  And leave enough files
unpacked to be able to work without Xpk in case of emergency.  Do not
distribute xpk-packed files, xpk is meant for private use (yet).

             CONTACT ADDRESS
             ---------------
Send congrats, suggestions, bug reports, flames (in that order) to

Christian Schneider
```

1.32 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 won't 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)
    won't 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're 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
  XPKSmart 1.0
  - first internal Release

  xSmart 1.0
  - 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
```

```
  xPack 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

  xPack 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

  xPack 1.2
  - added "PROGRAM" parameter
  - suffixes may be removed while unpacking

  xPack 1.3
  - wasted my time with a special function for handling ILBM-files
  - added "QUIET" parameter
  - "FORCE" is switched on automatically if a password is supplied.

  xPack 1.4
  - changed "FILE/M" to "FILE/M/A" in template
  - added "XSCAN" and "LOSSY" parameter

  xPack 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
```

1.33 xpk

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

       -m = packing method
       -f = force repack
       -s = don't 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
       don't remove the original.

  -r = recur. If any directories are encountered, they're 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


HISTORY
  xPK 1.0
  - First public release

  xPK 1.1
  - Docs written
  - Version string added

COPYRIGHT
  Freely distributable for noncommercial use.

AUTHOR
  Urban Dominik Müller
```

1.34 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.

HISTORY
  xQuery 1.0
  - First public release

  xQuery 1.1
  - Version string added
  - Docs added

COPYRIGHT
  Freely distributable for noncommercial use.

AUTHOR
  Urban Dominik Müller
```

1.35 xtype

```
SYNOPSIS
  xType filenames

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

EXAMPLE
  xType intuition.doc.nuke

HISTORY
```

```
  xType 1.0
  - First public release

  xType 1.1
  - This doc added
  - Version string added
  - No longer forgets open files when interrupted

COPYRIGHT
  Freely distributable for noncommercial use.

AUTHOR
  Urban Dominik Müller
```

1.36 xup

```
SYNOPSIS
  xUP [-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, stores uncompressed version
       under the same name minus .xpk suffix
  -p = password. Uses given password for decompression

EXAMPLE
  xUP -p secret mytext

HISTORY
  xUP 1.0
  - First public release

COPYRIGHT
  Freely distributable for noncommercial use.

AUTHOR
  Urban Dominik Müller
```

1.37 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'll 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'll
  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", DON'T 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's
  packed or not. That's why it's 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'll
  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 don't want to explain the format exactly, because people
      shouldn't 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
```

1.38 contacts

```
Please remember, that some of these addresses may be false, so don't
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

Other related persons:

  Harmut Goebel   Oberon Interface & examples
  Martin A. Blatter xDrop
  Matthias Meixner  xpkarchive.library, SHRI
  Kristian Nielsen  XFH
  Nicola Salmoria XFH commodity
  Matthias Scheler  XFH, xPack
  Christian Schneider XPK concept, xLoadSeg
  Markus Wild   GCC interface & examples
```

1.39 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
```


1.40 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
```

1.41 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
```

1.42 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
```

1.43 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
```

1.44 contact stephan fuhrmann

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

1.45 contact martin hauner

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

1.46 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

1.47 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/

1.48 contact jorma oksanen

Name: Jorma Oksanen
Address: Ratastie 5 A 3
    14200 TURENKI
    FINLAND
E-Mail: tenu@freenet.hut.fi
    tntbf@walli.uwasa.fi

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 martin a. blatter

Name: Martin A. Blatter
Address: Pfaffächerstr. 59
    CH-8913 Ottenbach
    Switzerland
E-Mail: blatter@amiga.physik.unizh.ch
    cbmvax!cbmehq!cbmswi!zethos!blatter

1.52 contact matthias meixner

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

1.53 contact kristian nielsen

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

1.54 contact nicola salmoria

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

1.55 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

1.56 contact christian schneider

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