xpkdisk

COLLABORATORS			
	TITLE : xpkdisk		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		October 27, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

xpkdisk

1.1 xpkdisk.guide

```
xpkdisk.device
**************

   This file documents xpkdisk.device 37.5.

   Copyright (C) 1993 by Olaf 'Rhialto' Seibert.

   Permission is granted to copy the whole package, in accordance with
the GNU General Public License, and some additional restrictions.


  Overview              What is in the package?
  Requirements          What else do you need?
  Usage                 How do you use it?
  Limitations           What does it not do?
  Implementation        Interesting things to know.
  License               What you may do with it.
  Index                 Where to find what in this manual.
```

1.2 xpkdisk.guide/Overview

```
Overview
********

   'xpkdisk.device' is an exec-style device that looks like
trackdisk.device and similar disks. The difference is that it compresses
its data and stores it in multiple files in an existing filesystem.

   * Now includes a utility to reclaim space from deleted files.

   * Now uses a hierarchical track naming scheme for faster access!
     See Upgrade on what to do if you are upgrading from the old naming
     scheme.
```

* Now behaves better in cace of threatening lack of disk space.

* Now handles read-only media.

It uses the XPK (eXternal PacKer) standard to do the actual compression. This has many advantages:

* You can choose which compression is most effective for your purpose. For one disk, you might want a very tight compression that is slow, for another you may prefer a faster but less efficient compression.

* As soon as another XPK library is released, you can use it for your compressed disk(s).

* Because each track is a standard XPK-compressed file, you can use the normal XPK utilities to uncompress your data, should you wish to do so.

* You can change the compression type of a disk on-the-fly. All already compressed data remains perfectly usable, but upon re-compression your new choice will be used.

* You can even manually decompress and recompress your disk (when xpkdisk is not accessing it, of course) with another compression method.

1.3 xpkdisk.guide/Requirements

Requirements
****

What you further need is the XPK user package. At the time of this writing, version 2.5 is current, and can be found (at least) on aminet ftp sites under the name 'Xpk25Usr.lha'. This package contains the necessary libraries for using the programs.

Running 'xpkdisk.device' probably needs lots of processor cycles and a hard disk to be enjoyable in any real sense. If you only have an unaccellerated Amiga, you should probably stick to the faster (and thus less-efficient) compressing methods.

'xpkdisk.device' runs under Kickstart versions 1.2 and higher, but the fancy preferences program xdPrefs and xdClear require 2.04 or higher. Some other features are also non-functional under 1.2/1.3.

If you want to recompile the source, you also need the XPK developer's package, named 'Xpk25Dev.lha'.

1.4 xpkdisk.guide/Usage

Usage
*****

1.5 xpkdisk.guide/Installation

Installation
============

   Installation can be performed by the Installer, or you can do it by hand. In that case, you must carry out the following steps.

  1. First, you need to copy ‘xpkdisk.device’ to your DEVS: directory. On my system, I would keep it in LOCAL:devs, and DEVS: is a multi-assign pointing to SYS:devs and LOCAL:devs.

  2. Then, you need to create a Mountlist entry and add it to your ‘DEVS:Mountlist’ file. An example in the 1.3 style is given.

  3. Finally, you need to make an assignment XPKD: to a directory where you want the compressed disk to be stored. ‘xpkdisk.device’ will create subdirectories as necessary.

  4. Before the first use you must format the disk. The following options are recommended, as far as they are available in your system version:

     Format drive XD0: name Compress0 QUICK FFS DIRCACHE

1.6 xpkdisk.guide/Mountlist

Mountlist

```
     /*
      *  XpkDisk Partition: Unit 0, Size 1 Mb
      */
     XD0:
             Device = xpkdisk.device
             Unit   = 0
             Flags  = 0
             Surfaces  = 1
             BlocksPerTrack = 64
```

```
            Reserved = 2
            Interleave = 0
            LowCyl = 0  ;  HighCyl = 31
            Buffers = 20
            BufMemType = 1
            DosType = 0x444F5301
            Mount = 0
     #
```

   The important parameters are:

'Unit'
     This is your choice of the xpkdisk unit. Since there are 8 units, you can use values from 0 to 7.

'BlocksPerTrack'
     This is used by 'xpkdisk.device' to determine how large the chunks of compression should be. So in this example with 64 blocks per track, the disk is compressed in chunks of 32 kilobytes each.

'LowCyl, HighCyl'
     This determines the size of your compressed disk. It is not necessary that LowCyl equals 0. The disk size in bytes can be calculated with 'Size = 512 * BlocksPerTrack * Surfaces * (HighCyl - LowCyl + 1)'.

'DosType'
     A DosType value of 0x444F5301 specifies FFS. If you have it, DCFS is probably better.

   The other parameters are simply the same values that would be used for "normal" disks.

1.7 xpkdisk.guide/Preferences

```
Preferences
===========
```

   Unlike xpkdisk.device, the preferences program ('xdPrefs') requires at least system 2.04.

```
  xdPrefs               Nice user-interface.
  File Format           How to set your preferences "by hand".
```

1.8 xpkdisk.guide/xdPrefs

```
'xdPrefs'
---------
```

The 'xdPrefs' program (1) opens a window when started from either the Workbench or the Shell. No special Workbench processing takes place, so ToolTypes have no effect.

All settings that can be changed with 'xdPrefs' may be changed at any time during operation of xpkdisk.device.

There are several gadgets and menus available.

Unit
: This slider determines which unit you are seeing and setting. If you drag it before clicking on either Save or Use, your settings will be lost.

Save
: Saves the currently visible settings in the permanent settings file. This file is located in the XPKD: directory. It also implies "Use". It does not quit the program.

Use
: Saves the currently visible settings in a temporary settings file. This file is located in the ENV: directory. The unit is notified that it should re-read its internal settings. It does *not* quit the program.

Quit
: This button *does* quit the program (and so will the close gadget).

  The reason that these three functions are separate is that I hate the fact that usually "Save" and "Use" imply "Quit". Especially here, when you may want to change the settings of several units in one go.

Compression
: This string gadget allows you to specify the compression you'd like to use. This string should be in the standard XPK format of ABCD(.nnn): a four letter name, optionally followed by a period and a number from 0 to 100 to specify the efficiency.

  Watch out: it allows you to specify compression methods that are not (currently) available. In that case, no compression will be used.

Timeout required
: This item is the first of the 3 gadgets relating to xpkdisk.device's behaviour when faced with the option of compression its internal buffers. Every time a modification is made to the internal buffers, a timeout is restarted. When the timeout expires, it is examined whether it is appropriate to do the compression.

  Checking this option means that the compression will not be done unless the timeout has expired.

Time
: This slider determines the time used for the timeout of the previous gadget. The actual timeout will be between this value

and half this value, depending on the actual sequence of events.

CMD_UPDATE required
Normally, a filesystem sends 'CMD_UPDATE' commands to xpkdisk.device if it has completed its current operations. Every time such a command is received, it is examined whether it is appropriate to do the compression.

Checking this option means that the compression will not be done unless this command has been received.

There are several combinations possible of 'Timeout' and 'CMD_UPDATE'.

None checked
Compression will be done every time a 'CMD_UPDATE' command is received, and every time the timeout expires. In this case it is probably best to take a long timeout, unless the overlying program is lax about sending the 'CMD_UPDATE' commands.

Only 'Timeout' checked
Compression will be done every time the timeout expires. 'CMD_UPDATE' commands are effectively ignored, since the occurrence of the timeout leaves no work to do for the 'CMD_UPDATE'.

Only 'CMD_UPDATE' checked
Compression will be done every time a 'CMD_UPDATE' command is received. Timeouts are effectively ignored, since the occurrence of the command leaves no work to do for the timeout.

Both 'Timeout' and 'CMD_UPDATE' checked
This is the default case. It means that even when a 'CMD_UPDATE' command is received, compression is delayed until the timeout expires. Not usually, the timeout could expire first, in which case the 'CMD_UPDATE' command will be waited for.

This is convenient in case new disk operations occur soon afterwards, which is often the case.

To recapitulate: There are two occasions at which compression is considered: after a timeout, and upon receiving a 'CMD_UPDATE'. Both events are noted, and then it is determined whether all required events have happened. If so, compression takes place (and the events are cleared). If not, no further action is taken at that time.

Safe write
This enables the use of the ripcord (see General Usage).

Cache size
This is the highest number of tracks that xpkdisk.device will keep in memory. Setting this to a higher value usally increases speed, but at the cost of (by default) 32 kilobytes of memory per track. If there are already the maximum number of tracks in memory, and

```
     access to another track is requested, one of the current tracks is
     selected to be replaced, compressing and writing it to disk if
     necessary.

   ---------- Footnotes ----------

   (1)  'xdPrefs' requires system 2.04 or newer.
```

1.9 xpkdisk.guide/File Format

```
File Format
-----------

   The file 'XPKD:XpkDisk<unit>.prefs' that is produced by the
preferences program looks like

     CMDUPDATE=1,DELAY=1,SAFE=0,CACHE=8,TIME=5,METHOD=BLZW.100

   These are the default settings, and correspond to the settings of the
gadgets with similar indications.

   It must be a single line of text, with the items separated by a
comma.

   The order in which the settings are given is not important, but the
full name is: no abbreviations are possible.

   Refer to the previous section for a detailed explanation of all
items.
```

1.10 xpkdisk.guide/General Usage

```
General Usage Information
=========================

   Once you have mounted your compressed disk, you can use it as a
normal AmigaDOS harddisk.

Disk-full handling
------------------

   There is a provision for disk-full conditions of the underlying file
system, but this can't be perfect. This means that disasters may still
happen if you make your compressed disk too large for the space that is
actually available. This is only functional if you checked "Safe write"
in the xdPrefs program.

   If for any currently in-use unit the "Safe write" option is checked,
'xpddisk.device' creates a "ripcord" file ('XPKD:Ripcord') that has at
least the size of two uncompressed tracks. Every time a track is
```

written out, the available size on 'XPKD:' is checked. If it falls below the same threshold (two uncompressed tracks), a warning requester is put up *if you are running 2.04+*. It gives you the following options:

Rip Cord
: The file 'XPKD:Ripcord' is deleted, in order to make space, and the operation proceeds. This option is always chosen under 1.2/1.3.

Try Anyway
: The operation proceeds regardless of the little available space. This may cause the write to fail.

Abort
: The operation is aborted: the track is not written out.

When the track is not written, or when an error occurs while trying to write it, the track is kept in memory, even if this means that huge amounts of memory will be used. It is also remembered that it needs to be written, so that retries will occur.

Also, xpkdisk will report a write error to the upper file system. Hopefully, this will only affect the file you are currently writing, but you may have a stroke of bad luck.

This means that as soon as you get the requester, you should abort whatever program is writing to the compressed disk. The spare space from the ripcord *should* allow the file system to keep its directory information consistent, but if you let the writing continue there will not even be space for that and you may end up with a corrupted file system anyway.

If aborting the writing program won't work, or not quick enough, the following strategy *may* limit the damage done. The first few times, click on the "Abort" button. This will (hopefully) propagate an error condition from 'xpkdisk.device', through the file system, to the application. When the application aborts, and the file system is cleaning up, click on "Try Anyway" or "Rip Cord".

If you are using 1.3, the choice "Rip Cord" is always made for you.

Read-only disks

If the underlying filesystem is read-only, then the compressed disk will also be read-only. If the read-only status changes, you may need to give a DiskChange command for the compressed disk, so that the file system will re-examine the disk.

Compressed floppies

Compressed floppy disks are currently not very practical, but they may be possible if you use a non-binding Assign for XPKD:, i.e.

```
Assign PATH XPKD: DF0:
```

However, you can then use only one unit, and you have to be *very* careful to do a 'DiskChange XD0:' every time you swap floppies.

1.11 xpkdisk.guide/Upgrade

```
Upgrading from a previous release
=================================
```

If you want to upgrade from a previous release that didn't have the hierarchical track naming scheme yet, the program 'xdName' will be useful to you.

First, determine the highest track number that needs to be converted. To be on the safe side, use the value of 'HighCyl' from the Mountlist, but you can also use the highest number that is actually in use.

Then, in the Shell, use the command

```
xdName track >script
```

where "track" is the track number you determined. This will create a script that will create the necessary new directories and rename all tracks. Then execute this script to do the actual conversion. You don't need to keep the script around:

```
execute script
delete script
```

Should you wish to convert back, use the '-r' flag. If you just want to know the file for a certain track, use 'xdName -n track'.

1.12 xpkdisk.guide/Free Space Recovery

```
Free Space Recovery
===================
```

or, On A Clear Disk You Can Seek Forever...

The normal Amiga filesystem does nothing special with files that you delete. The only thing it does is unlink the file from the directory structure. The advantage is that you can recover the file if nothing new has been written over it.

The disadvantage of this is that if you delete a file, your disk does not shrink.

To overcome this little problem, I have written a program, called 'xdClear' (1), that fills all unused sectors on the disk with zeros. The theory is that whole lots of zeroes would be very compressible and therefore reduce actual disk space usage to a minimum. Additionally, if

it is used on an xpkdisk, if a whole track is not in use it deletes the file containing it, thereby eliminating all disk usage for that track. ‘xdClear’ only works if you are using the normal Amiga filesystem.

Usage: xdClear <devicename>

for example ‘xdClear XD0’. You must give the device name, not the volume name. It does not matter if you include the colon (‘:’) in the name.

‘xdClear’ checks the bootblock for a value of "DOS\?", where \? may be anything, it calculates which block is supposed to be the file system root block, checks if its type is correct, checks the checksum, and finally sees if the bitmap is indicated as "valid".

‘xdClear’ scans the bitmap present on the indicated device, one track at a time. If a whole track is being unused, and the device is ‘xpkdisk.device’, then the the track file is deleted. Otherwise, the unused sectors are filled with zeroes. This is only written back to the disk if the sector wasn’t all zeroes to begin with. This should save a lot of compression effort in case the disk was relatively clean already.

Since the format of the bitmap should be, but isn’t, documented in the AmigaDOS manual, I had to reverse engineer this. I have tested this with the most complex bitmap structure possible according to what is being documented with a 10000-track (312 M) compressed disk.

---------- Footnotes ----------

(1) ‘xdClear’ requires system 2.04 or newer.

1.13 xpkdisk.guide/format of the bitmap

Format of the bitmap

This is an addition to the information on pages 336-338 of the 3rd edition of "The AmigaDOS Manual".

"Bitmap Block"

```
          0: Balance to 0 checksum
  1..SIZE-1: Longwords of flags indicating "available" (1),
             or "in use" (0) for each block.
             The bit for block #n is in longword 1 + n/32,
             bit number n mod 32 (bit 0 is the lsb).
```

The first 25 bitmap blocks are recorded in the root block, at longwords SIZE-49 .. SIZE-25. With the FFS, if more bitmap blocks are needed, they are recorded in a "bitmap extension block". The key of the first bitmap extension block is stored in rootblock longword SIZE-24.

"Bitmap Extension Block"

```
     0..SIZE-2: Sector numbers (i.e. keys) of bitmap blocks,
                or 0 if nonexistent.
        SIZE-1: Sector number of next bitmap extension block,
                or 0 if nonexistent.

     Note that bitmap extension blocks don't have a checksum.
```

1.14 xpkdisk.guide/Limitations

```
Limitations
***********

   * There is a provision for disk-full conditions of the underlying
     file system, but this can't be perfect. This means that disasters
     may still happen if you make your compressed disk too large for
     the space that is actually available. This is only functional if
     you checked "Safe write" in the xdPrefs program. (See
     General Usage and xdPrefs.)

   * There are only 8 units available. Since each unit in general
     requires a lot of cache memory, this should be plenty. However,
     nobody keeps you from using multiple partitions per device. Keep
     in mind then, that you need to have the same track size
     (BlocksPerTrack) for all partitions.

   * When a unit of 'xpkdisk.device' is opened, it scans the list of
     mounted devices to find out its track size. It just looks for its
     name and unit number and uses the first match. If not found, the
     track size defaults to 64 blocks. This means that if you *ever*
     intend to use a different track length for a unit, it must
     *always* be mounted before accessing it.

     This is not so bad as it seems, because specifying 'Mount = 0'
     costs almost no memory.
```

1.15 xpkdisk.guide/Implementation

```
Implementation
**************

How are all the tracks stored?
==============================

   Since version 37.3, each track of xpkdisk.device is stored in a file
with a name like 'XPKD:Unit1/Group_0000_HL/Track_001d_KE'.

   Why such a funny name? That is simple: names like these minimise the
number of disk accesses needed to find the file. For example, the file
```

header of this file can be retrieved in exactly 3 disk accesses. The same is true for all other tracks (as long as you have at most 72 * 72 = 5184 tracks; this equals 162 megabytes of uncompressed disk space). This works as follows.

Each unit of xpkdisk.device changes it current directory to 'XPKD:Unit<number>'. Then, when this track is requested, AmigaDOS looks up directory 'Group_0000_HL'. These directories each group (at most) 72 tracks, and as long as there are at most 72 directories, all directories have the a unique "hash value". Each directory block can store 72 pointers to contained files or directories, and the hash value determines in which of these slots the pointer should go. So if each track hashes uniquely, this means that the directory block can be found in a single disk access. (If the file or directory found at the location in the hash table is not the desired one, a chain is followed which means more disk accesses are needed.)

The number in the directory name is the hexadecimal representation of the first track in it.

The same is true in each directory: the names of the tracks are constructed such that each has a different hash value. In fact, the hash values in each directory neatly range from 0 for the lowest track to 71 for the highest track.

Compare this with the old situation (in version 37.2), where all tracks were stored in a single directory. Even if you have a modest disk of 20 megabytes, this means there are 640 tracks. On average, this would mean that 640 / 72 = 9 tracks would hash to the same hash table entry, which in turn would mean 1 + 4.5 (average) accesses to find a track, and 1 + 9 accesses worst case.

In reality the worst case was even worse, since the hash values were not evenly distributed. The longest hash chain would be 13, and the shortest 4. This makes the worst-case number of accesses 14, which is pretty bad.

This improvement in access speed comes at a small price: one extra disk block per 72 tracks: 9 extra blocks in the example 20 M case.

A note on track size
====================

Since file header blocks can store pointers to at most 72 data blocks, it is recommended that you make tracks not larger than 72 blocks.

1.16 xpkdisk.guide/License

License
****

1.17 xpkdisk.guide/Index

Index
*****

Unit	Mountlist
Upgrade	Upgrade
Usage	General Usage
xdClear	Free Space Recovery
xdName	Upgrade
xdPrefs	xdPrefs
Xpk25Dev.lha	Requirements
Xpk25Usr.lha	Requirements