

LTOOLS - Access your Linux files from Windows 9x/ME and Windows NT/2000/XP

by Werner Zimmermann

The LTOOLS provide under Windows a similar functionality as the MTOOLS do under Linux: They let you access your files on the "hostile" filesystem.

Using LTOOLS from the Command Line

At the heart of the LTOOLS is a set of command line programs, which can be called from DOS or from a DOS-Window in Windows 9x/ME or Windows NT/2000/XP. They provide the same functionality as the well-known LINUX commands 'ls', 'cp', 'rm', 'chmod', 'chown' and 'ln'. Thus, under DOS/Windows you can

- list Linux files and directories (command: ldir),
- copy files from Linux to Windows and vice versa (commands: lread, lwrite),
- delete or rename Linux files (commands: ldel, lren),
- create symbolic links (command: lln),
- create new Linux directories (command: lmkdir),
- modify a Linux file's access rights and owner (command: lchange),
- change the Linux default directory (command: lcd),
- set the Linux default drive (command: ldrive) and
- show your harddisk partition setup (command: ldir -part).

As with many UNIX tools, these functions are included in a single executable, which is called with a bundle of command line parameters. To make your life easier, a set of batch files (shell scripts) are provided, so that you don't need to remember and type in all these parameters.

Additionally there is a Unix/Linux version of the LTOOLS, so that you can use them under Solaris or even under Linux, when you want to access a file on another harddisk partition without mounting this partition.

LTOOLgui - a Java GUI for the LTOOLS

Command line programs are old fashioned! Where is LTOOLS graphical user interface? Well, no problem: Use LTOOLgui. LTOOLgui, written in Java using JDK 2's Swing library, provides a Windows Explorer like user interface (Fig. 1). In two sub-windows LTOOLgui shows your DOS/Windows and your Linux directory trees. Navigating can be done by the usual point-and-click actions. Copying files from Windows to Linux or vice versa can be done by copy-and-paste or by drag-and-drop. Clicking the right mouse button will open a dialog to view and modify file attributes like access rights, GID or UID. Double clicking on a file will start it, if it is a Windows executable, or open it with it's associated application. This even works with Linux files, if they have a registered Windows application.

BTW: You can also use LTOOLgui as a file manager under Linux. As the LTOOLS command line programs also come in a Linux version, thus you may access files on disks, without mounting them.

The author chose Java for LTOOLgui, because Java is especially suited for low level harddisk access ... only joking! No, of course, this is not possible in Java at all. If you want to access hardware directly, you have to use C++ code and JNI (Java to Native Interface). However, as the JNI only works for 32bit code, under Windows 9x/ME this would mean to use '32bit to 16bit thinking' (see below). As the author did not like the idea to combine Sun's Java with Microsoft's MASM code, he took another approach. He simply uses LTOOLS command line program, which get's called from Java via the well-known stdin/stdout- interface. So for the Java side, hardware access means simple stream based file I/O.



Fig. 1: Java-based LTOOLgui graphical user interface

File access over the Internet?

No doubt, any state of the art program must be Internet aware! Well, if you run LREADjav on a remote computer and you connect to it via LTOOLgui's connect button, you may access Linux files on this remote server as if they were local. LREADjav is a simple server daemon, which translates request, issued by LTOOLgui over TCP/IP, into LTOOLS command line program calls and sends the output of the command line programs back via

TCP/IP to LTOOLgui (Fig. 2). Of course, you can not only view directory listings but can do all remotely, what you can do locally, including file upload and download. The remote machine may run Unix/Linux or Windows. Today, this is more like a toy than a serious application, because LREADjav may pose security problems. In the default configuration, it can only be used from 'localhost', but it can be configured to allow connections from 3 different remote clients. But they are identified via their IP address only, there is no password protection or the like. However, if a user has a serious application for that, he can easily implement a login/password scheme ... It's all Open Source!

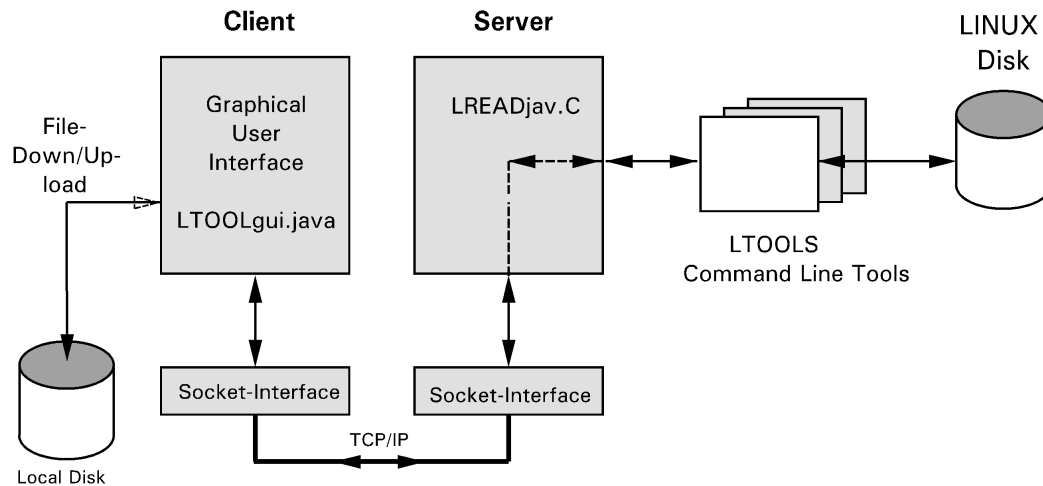


Fig. 2: LTOOLgui for remote access

No Java? Use your Web Browser!

Maybe you don't have Java 2 installed. Well, no problem, as long as you do have a web browser. Start 'LREADsrv' and your web browser and as URL type 'http://localhost' (Fig. 3). Now your Linux directory listing should show up graphically in your web browser. LREADsrv is a small local web server, which via a simple CGI-like interface makes the LTOOLS accessible via HTTP-requests and converts their output dynamically into HTML pages (Fig. 4). Of course, this does not only provide local access, but also allows remote access via the Internet. However, for remote users LREADsrv does have the same low level of security as LREADjav.

Because LREADsrv is based on HTML forms, which e.g. do not support drag-and-drop or direct copy-and-paste, working with your web browser is a little less convenient than working with the Java based GUI. Nevertheless it provides the same features.

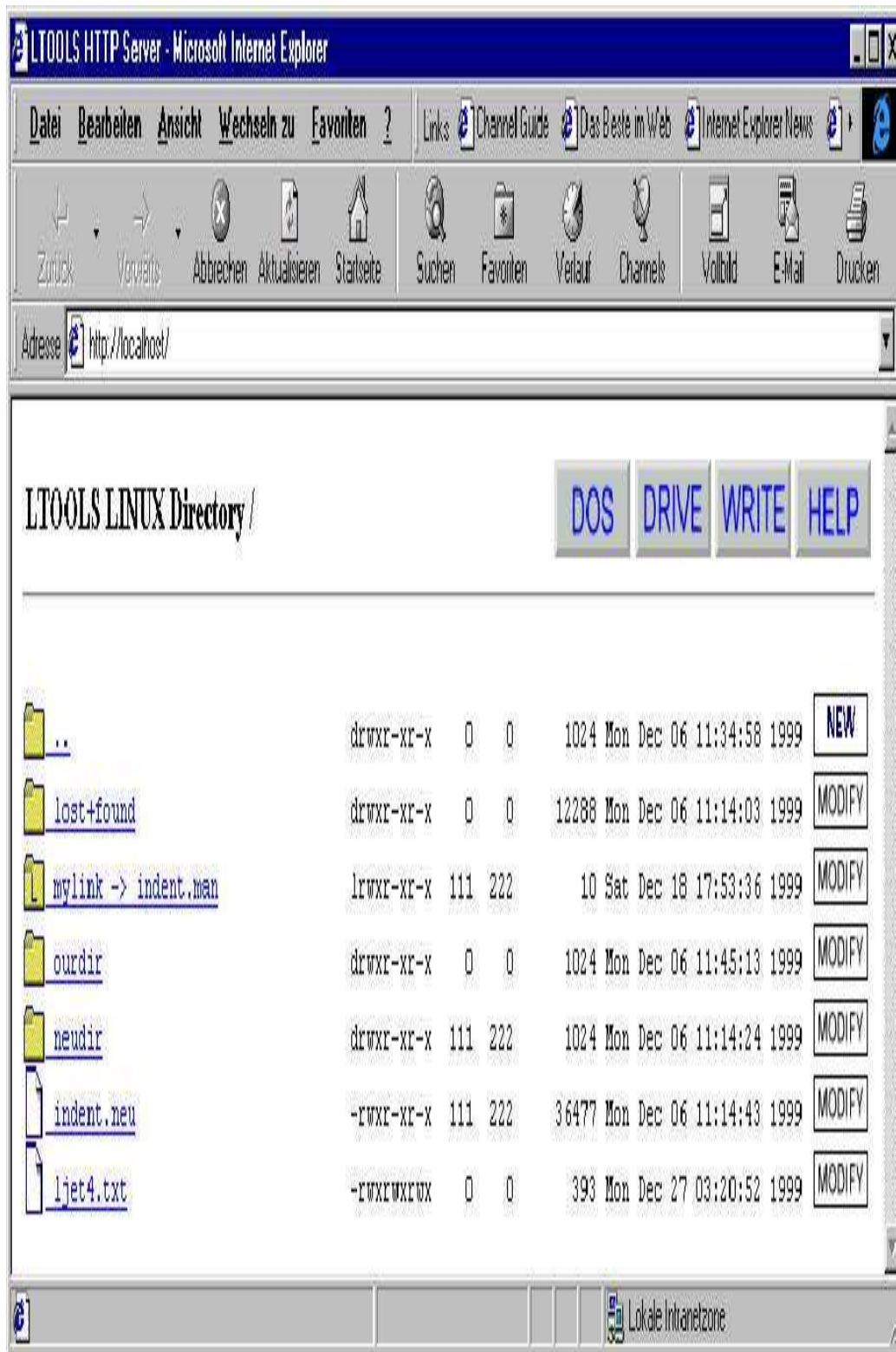


Fig. 3: Exploring Linux files with Microsoft's Internet Explorer

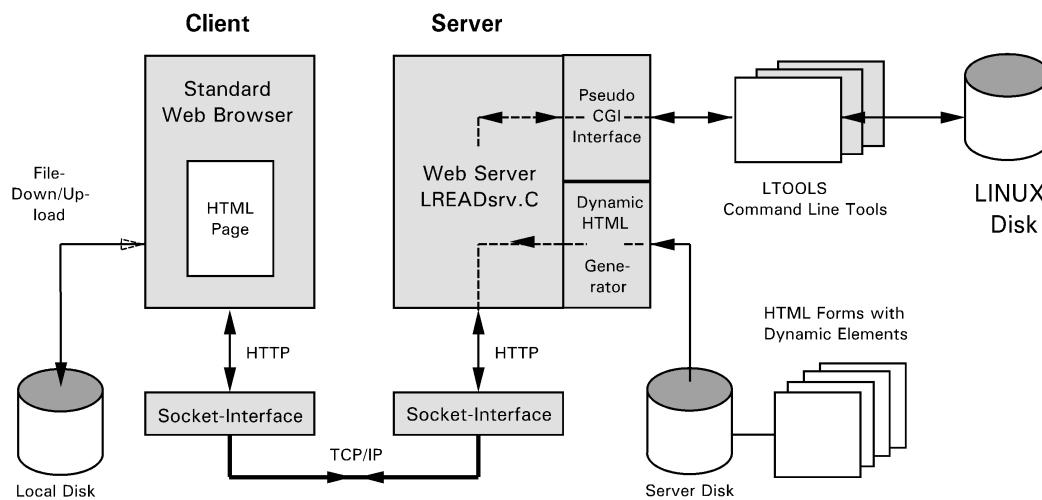


Fig. 4: LREADsrv - HTTP based access to Linux files

LTOOLS Internals - Accessings Harddisk under Windows

As DOS/Windows itself does not support interfaces to foreign filesystems, the LTOOLS must access the "raw" data bytes directly on the disk. To understand the internals of the LTOOLS, you need to have a basic understanding of the following areas:

- How harddisks are organized in partitions and sectors and how they can be accessed, i.e. how "raw" bytes can be read or written from disk. This information can be found e.g. in /2,3/.
- How Linux's Extended 2 filesystem is organized. A good overview about all the inodes, groups, blocks, bitmaps and directories stuff can be found e.g. in /4/.

This automatically leads to a layered architecture of the LTOOLS kernel (Fig. 5), which consists of several C files:

- The lowest layer 1 (in file Readdisk.c) physically accesses the harddisk. This layer deals with (nearly all) differences between DOS, Windows 9x/ME, Windows NT/2000/XP and Linux/Unix concerning direct harddisk access and tries to hide them from the higher layers. More about that soon.
- Layer 2 deals with the UNIX typical inode, block and group structures, into which the Extended 2 filesystem is organized.
- Layer 3 manages the directory structure of the filesystem.
- The highest layer 4 (in Main.c) provides the user interface and scans the command line parameters.

By scanning your harddisk's partition table, the LTOOLS try to find your first Linux partition on your first harddisk automatically. If you want to access another partition or disk, you have to specify it by command line parameter '-s', e.g. '-s/dev/hdb2'. Alternatively you may set another default drive and partition via command 'ldrive'. To find out, which partitions you have, call 'ldir -part'.

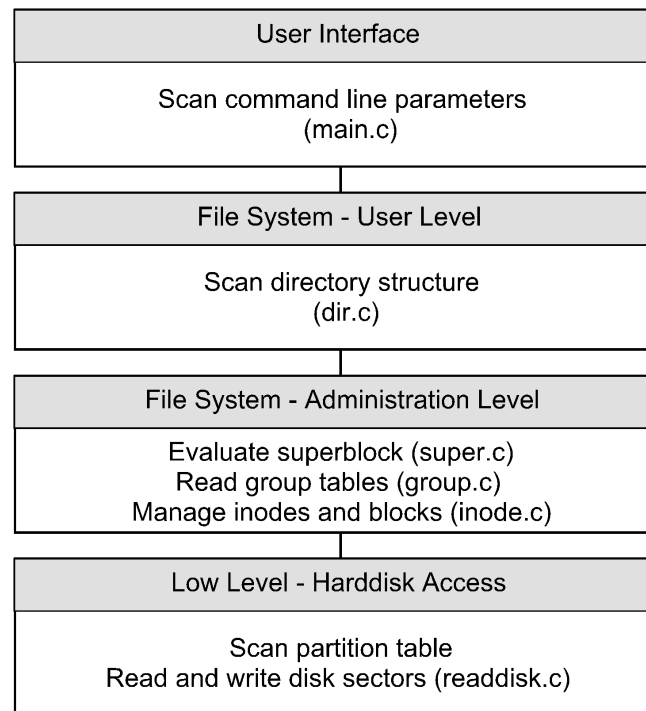


Fig. 5: LTOOLS layered architecture

Life was easy in the good old days of DOS. There was only one way for low-level read or write access to your harddisk: BIOS interrupt 13h /3/. BIOS data structures limited harddisks to 1024 cylinders, 63 heads and 255 sectors of 512 bytes, i.e. 8GB. Most C compilers provided a function named biosdisk(), so that this function could be directly used without needing to code in assembly language. To deal with bigger harddisks, some years ago 'extended' int 13h functions were introduced. To overcome the BIOS limitations, these functions use a linear addressing scheme, logical block addresses (LBA), rather than the old cylinder-head-sector (CHS) addressing.

This still works in Windows 9x/ME's DOS window (Table 1), at least for read access and as long as the program is compiled with a 16bit compiler. (The LTOOLS use Borland C, the Windows NT/2000/XP version also compiles with Microsoft Visual C, the Unix/Linux version uses GNU C). If you want low level write access, you need 'volume locks' /3/. This mechanism informs the operating system, that your program is performing direct disk writes bypassing the operating system drivers, so that Windows can prevent other programs from accessing the disk until you're done. Again this can be done without assembly programming by using the C compiler's ioctl() function.

In a 16bit Windows program BIOS functions can only be called via DPMI. As most C Compilers do not provide wrapper functions, this would require (inline) assembler. However, Win16 does not allow command line programs at all, so don't worry ...

In Windows NT/2000/XP's DOS box, using BIOS int 13h will lead to a GPF (General Protection Fault). Due to safety reasons, Windows NT/2000/XP does not allow direct harddisk access bypassing the operating system. However, Microsoft provides a solution, which is nearly as simple as what you would write under Unix/Linux:

```
int disk_fd = open("/dev/hda1", O_RDWR);
```

This would open your harddisk's partition /dev/hda1, to read you would call read(), to write you would call write(). Simple and straightforward, isn't it? Under Windows NT/2000/XP, if

you use the WIN32 API /5/, function CreateFile() does not only allow to create and open files, but also disk partitions:

```
HANDLE hPhysicalDrive = CreateFile("\\\\.\\PhysicalDrive0",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    0, OPEN_EXISTING, 0, 0 );
```

Reading and writing disk sectors can now be done via ReadFile() and WriteFile().

For a moment you might think, that you could use the same Win32 function under Windows 9x/ME. However, if you read on in the documentation for CreateFile(), you will find:

```
Windows 95: This technique does not work for opening a logical drive. In
Windows 95, specifying a string in this form causes CreateFile to return
an error.
```

Under Windows 9x/ME Microsoft's Win32 documentation recommends to call BIOS Int 13h via VWIN32, one of the system's VxDs (kernel drivers). If you try to do so, however, you won't succeed. Problem report Q137176 in Microsoft's Knowledge Base states, that - despite what the official Win32 documentation says - this does only work for floppy disks, not for harddisks. As the problem report says, for harddisks the only way is to call BIOS Int 16h in 16bit code. To call 16bit code from a 32bit program, you need Microsoft's "32bit to 16bit thunking"... This is not only another API (with other undocumented features or documented bugs?), thunking also requires Microsoft's thunking compiler, which from a definition script generates assembler code. From that a 16bit and a 32bit object file must be generated using Microsoft's assembler MASM. These will be linked with some dozend lines of C-code, which you have to write, resulting in a 16bit and a 32bit DLL (dynamic link library). By the way, you need not only 32bit Visual C++ for this, but you must also have an old 16bit version of Microsoft's C compiler... Got it? Using a bundle of proprietary, not widely used tools, would not be a good solution for an Open Source software tool like the LTOOLS!

Summarizing: There must be separate versions for DOS/Windows 9x/ME, Windows NT/2000/XP and Linux/Unix. To hide this from the user as far as possible, LTOOLS tries to find out, under which operating system it is running and automatically calls the appropriate executable.

Table 1: Low level harddisk access

Under DOS	Under Windows 9x/ME	Under Windows NT/2000/XP	Under LINUX/Unix
<ul style="list-style-type: none"> • BIOS Int 13h (need BIOS Extensions for disks above 8GB) 	<ul style="list-style-type: none"> • DOS programs: like DOS, but must use volume lock/unlock for write access • Win16 programs: must call BIOS Int 13h via DPMI 	<ul style="list-style-type: none"> • DOS programs: not allowed • Win16 programs: not allowed • Win32 programs: CreateFile(), 	<ul style="list-style-type: none"> • open(), read(), write()

	<ul style="list-style-type: none"> • Win32 programs: 32bit to 16bit thunking to a Win16 DLL 	ReadFile(), WriteFile()	
--	--	-------------------------	--

Safety concerns?

Yes, having the LTOOLS to a certain extend may pose security problems. Each user, who can run them, may access and modify files on the LINUX filesystem, e.g. change file access rights or file owners, exchange password files etc.. However, this is possible with a simple disk editor, too. Maybe, it's only a little more comfortable, when using the LTOOLS. Nevertheless, unlimited access is only possible, if running under DOS or Windows 9x/ME. Under Windows NT/2000/XP the LTOOLS user needs to have admin rights to access the harddisk directly. Under Unix/Linux in most standard installations also only the sys admin has access rights for the 'raw' disk devices /dev/hda, /dev/hda1, etc..

Are there any alternatives?

The LTOOLS are not the only solution for accessing Linux files from DOS/Windows. Probably Claus Tondering's Ext2tool /6/, a set of command line tools, developed in 1996, was the first solution for this problem. However, Ext2tool is restricted to read only access and does not run under Windows NT. Based on the Ext2tool, Peter Joot in 1997 wrote a windows NT version, still limited to read only /7/. Both programs were written in C, source codes are available.

John Newbiggin provides us with Explore2fs /8/, which comes with a very nice GUI and runs under Windows 9x and Windows NT. With its read and write access it provides the same features as LTOOLgui. BTW: John has done great work, because he managed to implement Microsoft's 32bit to 16bit thunking (see above) even under Borland's Delphi! As all Delphi programs Explore2fs integrates 'seamless' into Windows, but porting to non-Windows operating systems may be difficult.

History and Future

The first version of the LTOOLS was created under the original name 'lread' by Jason Hunter and David Lutz at the Willamette University, Salem/Oregon (USA). This first version ran under DOS, could show Linux directory listings and copy files from Linux to DOS and was limited to small IDE harddisks and LINUX on primary partitions.

The author took over maintenance and further development in 1996. Since then, the LTOOLS have learnt to deal with bigger harddisks, access SCSI drives, run under Windows 9x/ME and Windows NT/2000/XP, additional write access and were ported back to UNIX, to make them run under Solaris and Linux itself. They got a web browser based and a JAVA based graphical user interface etc. etc.. A lot of Linux users, most of them named in the source code, helped in testing and debugging. Thank you.

In the meantime, LTOOLS has reached version V4.7 /1/, maybe even more, when this article will be published. Besides additional features, a lot of bugs have been fixed - and most likely new ones have been introduced. A common problem has remained over the years: Nobody did foresee the rapid speed in harddisk technology, where disk sizes have exploded, which permanently hit operating system limits. Do you remember DOS's

problems with 512MB disks, Windows 3.x problems with 2GB partitions, BIOS's limit at 8GB and the various problems, which Windows NT does have at 2GB, 4GB and 8GB? It's only a moment ago! And by the way, even Linux has its problem: In kernels before 2.3, no file may exceed 2GB, as Linux like most 32bit Unix systems uses a signed 32bit offset pointer in `read()` or `write()` (this will be resolved in kernel 2.4 by changing offsets to 64bit values, but maintaining upward compatibility may drive Linux into the same problems as we discussed for Windows above). Software standardization for disk access always was much slower than the disk developers, so they invented proprietary solutions to overcome the operating system limits. And always the LTOOLS -and many other programmers - had to deal with it ... So don't be angry, if the LTOOLS don't work for you on your brand new 64GB drive. It's Open Source, so simply try helping to debug and further develop them!

And don't forget, if you use the LTOOLS: Do it at your own risk! Read only access to Linux is uncritical. However, if you use write access to delete files or modify file attributes on your Linux disk, the LTOOLS - and you as the user - can make a lot of nonsense. So always keep a backup!

References

1. <http://www.it.fht-esslingen.de/~zimmerma/software/ltools.html>: Homepage of the LTOOLS
2. Michael Tischer: PC-Intern 4. Data-Becker-Verlag
3. <http://www.cs.cmu.edu/afs/cs.cmu.edu/user/ralf/pub/WWW/files.html> Ralf Brown's interrupt list for x86-PCs
4. <http://metalab.unc.edu/pub/Linux/system/filesystems/ext2/Ext2fs-overview-0.1.ps.gz>: Gadi Oxman's overview about the Extended 2 filesystem.
5. Microsoft Windows Win32 API - Documentation, comes with most Windows C compilers or on the MSDN CDs
6. http://metalab.unc.edu/pub/Linux/system/filesystems/ext2/ext2tool_1_1.zip: Claus Tondering's Ext2tool
7. <http://metalab.unc.edu/pub/micro/pc-stuff/Linux/utils/dos/ext2nt.lsm>: Peeter Joot's Ext2nt
8. <http://uranus.it.swin.edu.au/~jn/linux/explore2fs.htm>: John Newbiggin's Explore2fs

About the Author

"In real life" Werner Zimmermann does teach control engineering, digital systems and computer architecture at the FH Esslingen - University of Applied Sciences, Esslingen, Germany. He has a hardware and software background in automotive and industrial embedded systems. His 'career' as a Linux system software developer started in 1994, when he purchased a CDROM drive, which was not supported by Linux ... So he developed 'aztcd.c', a Linux CDROM driver, which is still included in all standard Linux kernels, even if the drive now is very much outdated.