BDDE version 1.7e

J.L. Bezemer

14th June 2002

Contents

Chapter 1

User Guide

1.1 What is BDDE

BDDE (BetaDisk Dump Extractor) is an utility that enables a former BetaDisk user to transfer his files to the IBM-PC or Atari-ST without fiddling around with serial cables and COM-ports. The resulting files can be read into the Spectrum Emulators of Gerton Lunter, Kevin Phair (MS-DOS) and of Christian Gandler (Atari-ST). Apart from all this you can retrieve *every* bit of information concerning your BetaDisk. Even more than you ever could on your own Spectrum!

1.2 Who produced BDDE

BDDE is produced by J.L. Bezemer under the HanSoft-label. The author was quite active in the Spectrum scene from 1983 until 1987 and ported packages like Micro-Prolog, Artic Forth and HiSoft BASIC to BetaDisk. Apart from that he cracked several protection-schemes like LensLok and SpeedLok. He also wrote several distinctive menu-programs, the last being FRED (FRont End Display) that featured a graphical user-interface not unlike the Apple Macintosh, previously unknown in the Spectrum environment. It is included in this package.

1.3 Legal stuff

1.4 Requirements

For the conversion of a 40 tracks BetaDisk a 5.25" 360 kB drive is required. To convert a 80 tracks BetaDisk you need either a 3.5" 720 kB drive or a 5.25" 1.2 MB drive.

BDDE itself needs only a very modest configuration. Even a 128 kB machine with two 5.25" 360 kB drives will do in some circumstances. A rule of the thumb is, you'll need a machine with twice the online disk capacity of the size of the diskdump.

BDDE is available for MS-DOS, MS-Windows 3.x, Linux and Unix. Tested platforms include several 80286, 80386sx, 80386, 80486 MS-DOS machines, a Bull DPX/2 340 running BOS V2.00.69, a 486/33 clone running Coherent V4.2, an RS/6000 250 running AIX V3.2 (Unix), a Pentium III running Linux V2.2.14, a Pentium II running Windows V3.1 and a Pentium III running Windows 95 (Windows).

1.5 BDDE and media-conversion

There are three levels of conversion. The first just transfers the data from one medium to another, e.g. from harddisk to tape. This is called media-conversion. The second takes all data and transfers it from one file-format to another, e.g. when converting graphical data from a GIF-file into a BMP-file. This is called file-conversion. The final form of conversion is more difficult. It's called data-conversion. Both formats are not compatible so some data has to be added manually or "guessed". You'll find these conversions a lot when you're transferring data from a non-relational database into a relational database.

BDDE can do all file- and data-conversion, but is (still) unable to do media-conversion. Of course it can be done, but it takes a lot of low level programming. Worse, a solution can work on some hardware and refuse its services on other hardware. So I decided to leave that level of conversion alone and concentrate on file- and data-conversion, which is difficult enough.

That means that you have to get yourself a nice disk-analyzer, which can also dump the contents of the diskette examined on an MS-DOS (or Unix) readable disk. BDDE will then unpack these files for you. We are currently supporting "Anadisk" from Sydex and "ADU" from AME Computing Systems. Others might also be supported; just check your documentation on whether raw diskdumps are supported.

We have found two programs suitable for doing this job and we support them both. Neither BDDE nor its producer are connected in any way to the companies or person(s) who produce the programs we support. BDDE is an independent conversion-program. If you need support on the use of any of these products, you have to contact these companies or person(s).

1.6 Using BDDE

The conversion is done in two stages. First, the disk-analyzer converts the entire BetaDisk to a MS-DOS file. BDDE converts this (dump)file to several MS-DOS or Unix files, each holding the equivalent to a BetaDisk file. These files can either be further converted or read in without any conversion into a Spectrum Emulator. There are Unix, Linux, MS-DOS and Windows versions of BDDE. They produce exactly the same files.

1.7 Making the dump using AnaDisk

Read the instructions in ANADISK.DOC concerning ADCONFIG.EXE and configure Anadisk accordingly. When you're done, put the BetaDisk in either drive A: or B: (depends on the kind of computer you use and the disk you want to convert). Now start Anadisk and a menu will appear.

Use the cursor-keys to highlight DUMP and press <ENTER>. A next menu will appear. Use the cursorkeys again to highlight the A: or B: drive. Choose the drive where the BetaDisk is inserted. Don't press <ENTER> yet! Anadisk will ask you whether one side or both sides have to be dumped. Choose both sides (unless you're working with a single sided disk). Finally you're asked if you want to include sector-information.

The default is "No". You can use these dumps with BDDE, but you make your life a lot easier by answering "Yes" to this question. You can safely press <ENTER> now. BDDE uses this sector-information to check its position. A real drive does the same thing. That is, except for drives that position the head using special disks and diodes. These are called hard-sectored disks, but I haven't seen one since the Exidy Sorcerer (and that was a long time ago!).

We're almost there. Just enter the filename you want the dump written to (any valid MS-DOS filename will do) and press <ENTER>. Remember the drive must be able to hold all the data Anadisk writes to the dump-file (and please do not try to write a dump to a BetaDisk!). A dump from an 80 tracks, double sided disk (including sector-IDs) will result in a 660 kB file. Finally, you're asked which track to start from and which track to end the dump. If you're not sure, enter 0 and 79. If it's a 40 tracks disk Anadisk will adjust accordingly. But if you want to do it nice and clean enter 0 and 39 for 40 tracks disks..

Ok, sit back and relax. Anadisk will do the job for you. It will keep you posted with a pointer and messages (read/write, etc.). Don't let warning messages like a difference in Logical Side ID ruin your day. Everything is alright. When Anadisk is done, you can go back to the DOS-prompt by hitting the spacebar. Now let's put BDDE to work.

1.8 Making the dump using ADU

Read the instructions to ADU. When you're done, put the BetaDisk in either drive A: or B: (depends on the kind of computer you use and the disk you want to convert). Now start ADU and a prompt will appear.

If you want to convert the BetaDisk in drive A: enter:

```
L 0
```

Else enter:

```
L 1
```

If you want to convert a single sided, 40 tracks BetaDisk enter:

```
VB 16; VT 40; VZ 256; VH 1; VO 0; C <filename> 0/39; Q
```

The entry <filename> stands for the name of your dumpfile. If you want to convert a double sided, 40 tracks BetaDisk enter:

```
VB 16; VT 40; VZ 256; VH 2; VO 0; C <filename> 0/79; Q
```

If you want to convert a single sided, 80 tracks BetaDisk enter:

```
VB 16; VT 80; VZ 256; VH 1; VO 0; C <filename> 0/79; Q
```

If you want to convert a double sided, 80 tracks BetaDisk enter:

```
VB 16; VT 80; VZ 256; VH 2; VO 0; C <filename> 0/159; Q
```

That should do it. When ADU is done, you are returned to the DOS-prompt. Now let's put BDDE to work.

1.9 Putting BDDE to work

Let's do something very simple. Like displaying the help-screen. Type 'bdde' at the prompt and press <ENTER>. The following screen will be displayed (except when you're working with MS-Windows):

```
BetaDisk Dump Extractor V1.7e - (c) 1991,94  HanSoft & Partners

   Usage: BDDE <command> [options] [<filelist|@listfile> <directory>]

Commands: ls, df, di, cp, od, dc, sh
Options : -D%u    (Set debug level to %u; all)
          -i%u    (Set interleave to %u; all)
          -M%u    (Use mediatype %u; all)
          -F%s    (Mount BetaDisk dumpfile %s; all)
          -I%s    (Initialize using init-file %s; all)
          -l[-]   ([Do not] show a long directory listing; ls)
          -u[-]   ([Do not] use erased files too; cp, ls)
          -s[-]   ([Do not] force sector filesize on error; cp)
          -C%u    (Use conversion method %u; cp)
          -d%c    (Set first array to variable %c; cp)
          -f[-]   ([Do not] use same variable for all arrays; cp)
          -#      (Force numeric arrays; cp)
          -$      (Force character arrays; cp)
          -S[-]   ([Do not] use random seek; cp, od)
          -t%u    (Dump track %u; od)
          -c[-]   ([Do not] dump track in ascii; od)
          -E%s    (Use %s to start Spectrum Emulator; sh)
          -p%s    (Use %s as prompt; sh)
```

For those among you who know Unix there is not much special to see. Just think of BDDE as a very small Operating System. After I had written the low-level routines of BDDE the whole thing became very transparent. To me the higher level routines seemed to work with tracks and sectors. However, BDDE works only with dump-files. That is why it is portable. But let's take a closer look to this helpscreen.

- With 'bdde' we start our mini-OS.
- Then we'll have to enter the command we want to execute. Only 'ls', 'df', 'di', 'cp', 'od', 'dc' and 'sh' are available.

As a matter of fact, 'ls' is a Unix-DIR and 'cp' is a Unix-COPY. 'df' indicates how much free disk-space is available, not unlike FREE in 4DOS or CHKDSK. 'di' provides a lot of technical information, like DISKINFO in Norton Utilities V4.x, but is *not* a Unix command. Then there is 'od' which means Octal Dump and does about the same thing as the 'D' command of DEBUG. I know, it's strange but 'od' cannot do an octal dump. It can only dump in ASCII or hex. 'dc' is a tiny 'desk calculator' (that's why it is named 'dc') and can sometimes be useful. Finally, there is 'sh' which doesn't do anything at all. However, it provides a friendlier user-interface, and I've come to like it myself. Please note that all of these commands are case-sensitive. 'LS' will *not* work!

- You can slightly alter the operation of a command by adding several options. All options are prefixed by '-' instead of '/' like MS-DOS. And they are case-sensitive just like Unix.
- When using 'cp' you must add a target-directory like 'c:\temp' (DOS) or '/tmp' (Unix). You can even add a trailing separator to the directory-name. If you do not use 'cp' adding a target-directory is an error.

If you violate any of these rules or the order of the arguments BDDE will either show you a helpscreen or issue an error-message. So don't call me if that's the only thing you see all the time: you're definitely doing something wrong!

1.10 A sample session

In this section we will make a quick tour through the world of BetaDisk conversion. We included a sample dumpfile (made by ADU) to play around with. It's called "bdde.dmp". Extract the archive in any directory you want. I can't guarantee that it will still be a .ZIP file since some sysops will repack the file with their favorite packer.

Once you've successfully extracted all files (including "bdde.dmp") enter:

```
bdde sh
```

Or just start it when you're working with MS-Windows. The MS-Windows version of BDDE doesn't require the 'sh' command. BDDE will now answer with:

```
$
```

Yes, it looks like standard Unix! For all you MS-DOS and MS-Windows users: "$" is the Unix way of saying:

```
C:>
```

Before BDDE can do anything you have to let it know what you're working with. If you're using a sector-headerless Anadisk dumpfile or an ADU dumpfile the option '-M0' will apply. If you're working with a Anadisk dump *with* sector-headers, you'll have to use the option '-M1'. Furthermore, you'll have to specify the dumpfile that you are using. Both absolute and relative pathnames are supported, but relative names will fail when you're changing directories.

Now we have to mount "bdde.dmp". BDDE will then know which file you're working with and won't ask for it again. One way to do this is by using the system-variable support BDDE provides. It's very easy to use. E.g. if your 'bdde.dmp' is located in the '/home/john' directory of your Unix-system enter:

```
BDDEINIT='-M0 -F/home/john/bdde.dmp'; export BDDEINIT
```

Likewise, if your BDDE.DMP file is located in the C:\TEMP directory of you MS-DOS system enter:

```
set BDDEINIT=-M0 -Fc:\temp\bdde.dmp
```

But you have to issue that command *before* you start up BDDE! Now it's to late for that. However, BDDEs shell has an alternative way of doing that. You can check whether a file is mounted by typing:

```
mount
```

BDDE will probably answer:

```
Nothing mounted
```

If you've forgotten what your working directory is, just type:

```
pwd
```

Which is also a Unix-command meaning "Present Working Directory". Let's assume you're working with MS-DOS and BDDE returns:

```
c:\temp
```

Now type:

```
mount -M0 -Fc:\temp\bdde.dmp
```

The file is mounted now and you can start your tour with a minimum of key-strokes. Please remember, that we will assume in the next sections that you're still working with the shell! Have fun!

1.11 Let's 'ls' a little

We can read what is on the disk by typing this command:

```
ls -l
```

A list not unlike this will appear:

```
                Addr    Len     B/D  First     First  Last        First  Last
Filename  Type  Len(B)  Run(C)  NEG  Tk/Sd/Sc  Abs.S  Abs.S  #Sc  Tk/Sc  Tk/Sc
==========================================================================
index     BASIC   9232    9232  240    1/0/01     16     52   37   1/00   3/04
FRED      CODE   59769   64300    0    3/0/06     53     75   23   3/05   4/11
keynes    BASIC  13223   13166   89    4/0/13     76    127   52   4/12   7/15
bt,ks&ei  BASIC   2620    2620  196    8/0/01    128    138   11   8/00   8/10
probe     BASIC  10143   10101   97    8/0/12    139    178   40   8/11  11/02
musicflr  BASIC   6900    6900   12   11/0/04    179    205   27  11/03  12/13
musicode  CODE   65333       0    0   12/0/15    206    206    1  12/14  12/14
flagrec   BASIC   1542    1298  250   12/0/16    207    213    7  12/15  13/05
MakeDir (3): Erased file '?eynsdoc' found: skipped
MakeDir (3): Erased file '?lide' found: skipped
HanSoft   CODE   16384       0    0   17/0/15    286    312   27  17/14  19/08
C         CODE   16384       0    0   19/0/10    313    339   27  19/09  21/03
I wish..  CODE   16384       0    0   21/0/05    340    366   27  21/04  22/14
birthday  CODE   16384    1627    0   22/0/16    367    393   27  22/15  24/09
slide     BASIC   4152    4038  200   24/0/11    394    410   17  24/10  25/10
<0>       DATA   33150     317  195   25/0/12    411    412    2  25/11  25/12
```

Left the BetaDisk filename. The next column shows the file type. The next two columns are only interesting when it's a CODE file. The Addr-column indicates the loading address, the next one indicates the starting address, that is when you own a BetaDisk V3 interface. When using BetaDisk+ it indicates the length of the CODE files. CODE files cannot autoRUN on a BetaDisk+ as a consequence.

The B/D NEG column doesn't contain very interesting information for the average user. It was added to enable me to check the allocation algorithm. The First Tk/Sd/Sc column shows on which physical track/side/sector a file starts. Only interesting when you want to peek into a BetaDisk using Anadisk (see ANADISK.DOC). BDDE has a similar function. The First Abs.S column shows on which absolute sector a file starts (the very first sector is absolute sector 0 and so on). The Last Abs.S column indicates what sector is the last absolute sector which still belongs to the file. The #Sc column shows how many sectors the file spans. Finally the First Tk/Sc and Last Tk/Sc columns. This notation is used by TR-DOS itself. It indicates the first and last logical track and sector of the file. Both tracks and sectors are starting with 0. When you want to use the 'od' command the track number is important.

Not everything seemed to go *that* well. We've got two warnings on our hands. That is because BDDE found two erased files on our dump, which *CAN* be extracted! If you want to see them just add the '-u' option to your command-line:

```
ls -l -u
```

The ’&’ indicates that the file has been erased (unless you put that ’&’ there yourself).

If you do not use the ’-l’ option BDDE will only show you the names of the files on disk. This can be very useful as we will show you later on.

1.12 ‘df‘ and ‘di‘

These two commands provide you with information on the BetaDisk as a whole. I added these commands to allow users to obtain every possible bit of information without returning to the BetaDisk environment. Some information can’t even be obtained in this environment. Just type:

df

And the following list will appear:

```
Volume label is: BDDE test
    Password is:

160 kb,  (640 sectors) total
 99 kb,  (397 sectors) in 16 files, 2 deleted
  4 kb,  (16 sectors) in directory
 56 kb,  (227 sectors) free
```

It will show you the name of the disk, its password (Sure, when you’re using BetaDisk V3 or below), and some other information on the disk usage. ‘di‘ gives you quite another kind of information:

```
di
PHYSICAL DISK INFORMATION               LOGICAL DISK INFORMATION
-------------------------               -------------------------
Format....  40 tracks, single sided     Media descriptor.............  19h
Heads.........................   1      Total sectors................  640
Cylinders.....................  40      Bytes per sector.............  256
Starting head.................   0      First sector of directory.....    0
Starting cylinder.............   0      Number of sectors on directory..  16
Starting sector...............   1      Maximum number of dir. entries..  128
Ending head...................   0      First sector of data area.....   16
Ending cylinder...............  39
Ending sector.................  16
```

Very technical indeed. When you don’t know what to do with it, just forget it. It’s not very important when you’re only interested in converting files. May be you can do something with the Format-entry. It indicates the disk-type. Yep, and it’s correct!

1.13 Getting down to business: ‘cp‘

May be you think: well, that’s all very nice but when do I get the hack my files converted to my machine. Slow down, it’s all there. Just use ‘cp‘. Type:

```
cp * .
```

All BetaDisk files are now copied to the default directory. Do you want to use another directory or drive?

```
cp * c:\emul
```

The files will now be copied to the C:\EMUL directory. However, do *not* try to read these files into any other emulator than the one from Gerton Lunter. It won't work! Oh no, not another conversion.. You won't need one. If you're converting files to your Atari St you only have to add an option like:

```
cp -C2 * c:\emul
```

The same applies when you're using the SpecEm emulator, only then you have to use the -C3 option.

When you add -C0 all the files will be copied, but without a header. The emulators need these headers to see what file is coming. Files without a header can be very useful when you want to convert them into something else, e.g. Tasword II files to MS-DOS or Unix-files. BDDE comes with a small utility to do just that, by the way. The option -C1 is default and makes files for the emulator of Gerton Lunter. If you are using V2 of his fine emulator you can also use the -C4 option which creates .TAP files. And I find them a lot easier to handle!

By the way, if you are using SMARTDRV.EXE V4 (or another cache-program with write-caching) this tip is for you! BDDE generates files very rapidly and (especially when you are using a large write-cache) sometimes it may seem that nothing is happening. Don't reset! You'll ruin all your work because resetting throws away the contents of the write cache. Your computer is just VERY busy writing all these files to disk. Just wait a minute so SMARTDRV can catch its breath again. Of course you can disable the write-cache, but that isn't really necessary.

Now the story tends to get a little different for Atari-ST and MS-DOS users. Most current users are MS-DOS users, so we'll tell them first how it's done. We'll come back to you. SpecEm users, please jump to section1.13.3. Atari users, please continue with section1.13.2.

1.13.1 Z80

The complete command line for the users of Gerton Lunters emulator now reads:

```
cp * .
```

The next messages will be shown on your screen (that is, if you haven't forgotten to erase previously generated files):

```
(WARN ) MakeDir: Erased file '?eynsdoc' found: skipped
(WARN ) MakeDir: Erased file '?lide' found: skipped
(INFO ) RunLine: Program 'index' runs from line 10000
(INFO ) MakeCopy: BetaDisk file 'index' copied to '.\index.bas'
(INFO ) MakeCopy: BetaDisk file 'FRED' copied to '.\fred.cod'
(INFO ) RunLine: Program 'keynes' runs from line 200
(INFO ) MakeCopy: BetaDisk file 'keynes' copied to '.\keynes.bas'
(INFO ) RunLine: Program 'bt,ks&ei' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'bt,ks&ei' copied to '.\btks&ei.bas'
(INFO ) RunLine: Program 'probe' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'probe' copied to '.\probe.bas'
(INFO ) RunLine: Program 'musicflr' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'musicflr' copied to '.\musicflr.bas'
(INFO ) MakeCopy: BetaDisk file 'musicode' copied to '.\musicode.cod'
(INFO ) RunLine: Program 'flagrec' runs from line 10
(INFO ) MakeCopy: BetaDisk file 'flagrec' copied to '.\flagrec.bas'
```

```
(INFO ) MakeCopy: BetaDisk file 'HanSoft' copied to '.\hansoft.cod'
(INFO ) MakeCopy: BetaDisk file 'C' copied to '.\c.cod'
(INFO ) MakeCopy: BetaDisk file 'I wish..' copied to '.\iwish.cod'
(INFO ) MakeCopy: BetaDisk file 'birthday' copied to '.\birthday.cod'
(INFO ) RunLine: Program 'slide' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'slide' copied to '.\slide.bas'
(INFO ) GetAType: '<0>' (317 bytes) contains character array A (312 elem. in 2 dims.)
(INFO ) MakeCopy: BetaDisk file '<0>' copied to '.\0.chr'
```

Every error message is shown in this format. First of all the errorlevel, then the C-function that generated the message - in this case MakeCopy() and RunLine() - and finally the message itself.

One thing is very obvious: BDDE filters out all characters that cannot be part of a filename under DOS. Then the whole name is converted to lower case even when you're working with the Unix-version. Lowercase filenames are also very convenient when you're working with PC-NFS. If there are no characters in the BetaDisk filename that are allowed under DOS, the name 'noname' is assigned to this file.

In case there would be a conflict between two filenames, e.g. 'STRIPOKR' and 'stripokr' which are two perfectly different filenames to TR-DOS, BDDE will still try to create two different filenames. If BDDE were a dumb program you could never retrieve the first file, because it would be overwritten by 'stripokr'. However, BDDE takes notice of the file on disk and changes the extension of 'stripokr'. If both are BASIC files the first one created would be called 'stripokr.bas' and the second 'stripokr.ba0'. And it continues to do so until 'stripokr.ba9' has been written. Then BDDE has got one other trick up its sleeve. It starts to generate files from "bddeAA.AA0" to "bddeZZ.ZZZ". That's about 36^5 files. Most users don't even have that many files on disk.. And after that? Sorry..

Finally the extension indicates what kind of file has been extracted. The extension '.cod' is assigned to CODE files, the extension '.bas' to BASIC files and the extension '.nmb' or '.chr' to DATA files.

The old Spec has got to know a little more to read a file using the RS232 port. That's also true in the emulated version. That's why BDDE automatically generates the necessary 9 byte header. The header tells the Spectrum what kind of file is being read, how long it is and where it's got to be loaded. Well, when reading BASIC and DATA files it ignores the starting address, but *not* when reading CODE files! It's all explained in the source.

If you are currently using version 2.x of Gertons Z80 emulator you can also use .TAP file conversion. To activate it, just add the '-C4' option:

```
cp -C4 * .
```

The next messages will be shown on your screen (that is, if you haven't forgotten to erase previously generated files):

```
(WARN ) MakeDir: Erased file '?eynsdoc' found: skipped
(WARN ) MakeDir: Erased file '?lide' found: skipped
(INFO ) RunLine: Program 'index' runs from line 10000
(INFO ) MakeCopy: BetaDisk file 'index' copied to '.\index.tap'
(INFO ) MakeCopy: BetaDisk file 'FRED' copied to '.\fred.tap'
(INFO ) RunLine: Program 'keynes' runs from line 200
(INFO ) MakeCopy: BetaDisk file 'keynes' copied to '.\keynes.tap'
(INFO ) RunLine: Program 'bt,ks&ei' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'bt,ks&ei' copied to '.\btks&ei.tap'
(INFO ) RunLine: Program 'probe' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'probe' copied to '.\probe.tap'
(INFO ) RunLine: Program 'musicflr' runs from line 1
```

```
(INFO ) MakeCopy: BetaDisk file 'musicflr' copied to '.\musicflr.tap'
(INFO ) MakeCopy: BetaDisk file 'musicode' copied to '.\musicode.tap'
(INFO ) RunLine: Program 'flagrec' runs from line 10
(INFO ) MakeCopy: BetaDisk file 'flagrec' copied to '.\flagrec.tap'
(INFO ) MakeCopy: BetaDisk file 'HanSoft' copied to '.\hansoft.tap'
(INFO ) MakeCopy: BetaDisk file 'C' copied to '.\c.tap'
(INFO ) MakeCopy: BetaDisk file 'I wish..' copied to '.\iwish.tap'
(INFO ) MakeCopy: BetaDisk file 'birthday' copied to '.\birthday.tap'
(INFO ) RunLine: Program 'slide' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'slide' copied to '.\slide.tap'
(INFO ) GetAType: '<0>' (317 bytes) contains character array A (312 elem. in 2 dims.)
(INFO ) MakeCopy: BetaDisk file '<0>' copied to '.\0.tap'
```

Every error message is shown in this format. First of all the errorlevel, then the C-function that generated the message - in this case MakeCopy() and RunLine() - and finally the message itself.

One thing is very obvious: BDDE filters out all characters that cannot be part of a filename under DOS. Then the whole name is converted to lower case even when you’re working with the Unix-version. Lowercase filenames are also very convenient when you’re working with PC-NFS. If there are no characters in the BetaDisk filename that are allowed under DOS, the name ’noname’ is assigned to this file.

In case there would be a conflict between two filenames, e.g. ’STRIPOKR’ and ’stripokr’ which are two perfectly different filenames to TR-DOS, BDDE will still try to create two different filenames. If BDDE were a dumb program you could never retrieve the first file, because it would be overwritten by ’stripokr’. However, BDDE takes notice of the file on disk and changes the first character of ’stripokr’. If both are BASIC files the first one created would be called ’stripokr.tap’ and the second ’0tripokr.tap’. And it continues to do so until ’9tripokr.tap’ has been written. Then BDDE has got one other trick up its sleeve. It starts to generate files from "bddeAA.AA0" to "bddeZZ.ZZZ". That’s about 36^5 files. Most users don’t even have that many files on disk.. And after that? Sorry..

The old Spec has got to know a little more to read a file using the cassette-port. That’s also true in the emulated version. That’s why BDDE automatically generates the necessary 17 byte header. The header tells the Spectrum what kind of file is being read, how long it is and where it’s got to be loaded. Well, when loading BASIC and DATA files the starting address isn’t very important, but it is when loading CODE files! The filename is also included in the header. It’s all explained in the source.

I think I’ve spend enough time with you MS-DOS junkies. Just continue to section 1.14. I’m now going to tell the Atari-ST users how it’s done.

1.13.2 Atari

You found this section? Great, then we’ll continue. The complete command line for Atari-ST users reads:

```
cp -C2 * .
```

The next messages will be shown on your screen (that is, if you haven’t forgotten to erase previously generated files):

```
(WARN ) MakeDir: Erased file '?eynsdoc' found: skipped
(WARN ) MakeDir: Erased file '?lide' found: skipped
(INFO ) RunLine: Program 'index' runs from line 10000
(INFO ) MakeCopy: BetaDisk file 'index' copied to '.\index'
(INFO ) MakeCopy: BetaDisk file 'FRED' copied to '.\fred'
(INFO ) RunLine: Program 'keynes' runs from line 200
```

```
(INFO ) MakeCopy: BetaDisk file 'keynes' copied to '.\keynes'
(INFO ) RunLine: Program 'bt,ks&ei' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'bt,ks&ei' copied to '.\btks&ei'
(INFO ) RunLine: Program 'probe' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'probe' copied to '.\probe'
(INFO ) RunLine: Program 'musicflr' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'musicflr' copied to '.\musicflr'
(INFO ) MakeCopy: BetaDisk file 'musicode' copied to '.\musicode'
(INFO ) RunLine: Program 'flagrec' runs from line 10
(INFO ) MakeCopy: BetaDisk file 'flagrec' copied to '.\flagrec'
(INFO ) MakeCopy: BetaDisk file 'HanSoft' copied to '.\hansoft'
(INFO ) MakeCopy: BetaDisk file 'C' copied to '.\c'
(INFO ) MakeCopy: BetaDisk file 'I wish..' copied to '.\iwish'
(INFO ) MakeCopy: BetaDisk file 'birthday' copied to '.\birthday'
(INFO ) RunLine: Program 'slide' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'slide' copied to '.\slide'
(INFO ) GetAType: '<0>' (317 bytes) contains character array A (312 elem. in 2 dims.)
(INFO ) MakeCopy: BetaDisk file '<0>' copied to '.\0'
```

Every error message is shown in this format. First of all the errorlevel, then the C-function that generated the message - in this case MakeCopy() and RunLine() - and finally the message itself.

One thing is very obvious: BDDE filters out all characters that cannot be part of a filename under DOS. Then the whole name is converted to lower case even when you're working with the Unix-version. Lower-case filenames are also very convenient when you're working with PC-NFS. If there are no characters in the BetaDisk filename that are allowed under DOS, the name 'noname' is assigned to this file.

In case there would be a conflict between two filenames, e.g. 'STRIPOKR' and 'stripokr' which are two perfectly different filenames to TR-DOS, BDDE will still try to create two different filenames. If BDDE were a dumb program you could never retrieve the first file, because it would be overwritten by 'stripokr'. However, BDDE takes notice of the file on disk and changes the last character of 'stripokr'. If both are BASIC files the first one created would be called 'stripokr' and the second 'stripok0'. And it continues to do so until 'stripok9' has been written. Then BDDE has got one other trick up its sleeve. It starts to generate files from "bddeAA.AA0" to "bddeZZ.ZZZ". That's about 36^5 files. Most users don't even have that many files on disk.. And after that? Sorry..

The old Spec has got to know a little more to read a file using the cassette-port. That's also true in the emulated version. That's why BDDE automatically generates the necessary 17 byte header. The header tells the Spectrum what kind of file is being read, how long it is and where it's got to be loaded. Well, when loading BASIC and DATA files the starting address isn't very important, but it is when loading CODE files! The filename is also included in the header. Christian Gandlers emulator insists that the physical filename *and* the name in the header are identical. That's the only reason why BDDE produces slightly different filenames for the Atari-ST. It's all explained in the source.

So I guess you're going to be quite busy now on this machine with no buttons to click. In the meanwhile I got the time to explain the SpecEm fans a thing or two. Please continue with section 1.14.

1.13.3 SpecEm

You found this section? Great, then we'll continue. The complete command line for SpecEm users reads:

```
cp -C3 * .
```

The next messages will be shown on your screen (that is, if you haven't forgotten to erase previously generated files):

```
(WARN ) MakeDir: Erased file '?eynsdoc' found: skipped
(WARN ) MakeDir: Erased file '?lide' found: skipped
(INFO ) RunLine: Program 'index' runs from line 10000
(INFO ) MakeCopy: BetaDisk file 'index' copied to '.\index___.__b'
(INFO ) MakeCopy: BetaDisk file 'FRED' copied to '.\fred____.__c'
(INFO ) RunLine: Program 'keynes' runs from line 200
(INFO ) MakeCopy: BetaDisk file 'keynes' copied to '.\keynes__.__b'
(INFO ) RunLine: Program 'bt,ks&ei' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'bt,ks&ei' copied to '.\btks_ei_.__b'
(INFO ) RunLine: Program 'probe' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'probe' copied to '.\probe___.__b'
(INFO ) RunLine: Program 'musicflr' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'musicflr' copied to '.\musicflr.__b'
(INFO ) MakeCopy: BetaDisk file 'musicode' copied to '.\musicode.__c'
(INFO ) RunLine: Program 'flagrec' runs from line 10
(INFO ) MakeCopy: BetaDisk file 'flagrec' copied to '.\flagrec_.__b'
(INFO ) MakeCopy: BetaDisk file 'HanSoft' copied to '.\hansoft_.__c'
(INFO ) MakeCopy: BetaDisk file 'C' copied to '.\c_______.__c'
(INFO ) MakeCopy: BetaDisk file 'I wish..' copied to '.\iwish___.__c'
(INFO ) MakeCopy: BetaDisk file 'birthday' copied to '.\birthday.__c'
(INFO ) RunLine: Program 'slide' runs from line 1
(INFO ) MakeCopy: BetaDisk file 'slide' copied to '.\slide___.__b'
(INFO ) GetAType: '<0>' (317 bytes) contains character array A (312 elem. in 2 dims.)
(INFO ) MakeCopy: BetaDisk file '<0>' copied to '.\0_______.__r'
```

Every error message is shown in this format. First of all the errorlevel, then the C-function that generated the message - in this case MakeCopy() and RunLine() - and finally the message itself.

One thing is very obvious: BDDE filters out all characters that cannot be part of a filename under DOS. Then the whole name is converted to lower case even when you're working with the Unix-version. Lowercase filenames are also very convenient when you're working with PC-NFS. If there are no characters in the BetaDisk filename that are allowed under DOS, the name 'noname' is assigned to this file.

In case there would be a conflict between two filenames, e.g. 'STRIPOKR' and 'stripokr' which are two perfectly different filenames to TR-DOS, BDDE will still try to create two different filenames. If BDDE were a dumb program you could never retrieve the first file, because it would be overwritten by 'stripokr'. However, BDDE takes notice of the file on disk and changes the first character of 'stripokr'. If both are BASIC files the first one created would be called 'stripokr.__b' and the second '0tripokr.__b'. And it continues to do so until '9tripokr.__b' has been written. Then BDDE has got one other trick up its sleeve. It starts to generate files from "bddeAA.AA0" to "bddeZZ.ZZZ". That's about 36^5 files. Most users don't even have that many files on disk.. And after that? Sorry..

The old Spec has got to know a little more to read a file using the cassette-port. That's also true in the emulated version. That's why BDDE automatically generates the necessary 17 byte header. The header tells the Spectrum what kind of file is being read, how long it is and where it's got to be loaded. Well, when loading BASIC and DATA files the starting address isn't very important, but it is when loading CODE files! The filename is also included in the header. Kevin Phairs emulator insists that the physical filename *and* the name in the header are identical. That's the only reason why BDDE produces slightly different filenames for the SpecEm. It's all explained in the source.

1.14 Conditional extraction

But may be you don't want to extract all the files, but only the files that make up the 'musicfiler' program. You can do that with this version of BDDE! Just type:

```
cp musicflr musicode .
```

Don’t forget the ’-Cx’ option if you need it! BDDE will now only extract these two files. You may specify as many files as your Operating System allows, but please remember that only the original BetaDisk filenames are recognized. Issuing:

```
cp musicflr.bas musicode.cod .
```

will *not* work. However wildcards will! At least ’*’ will when you’ve placed it left, right or on both sides. BDDE will ignore it when it’s placed in the middle, but you can probably live with that. So these will work:

```
cp music* .
cp *e .
cp *ey* .
cp music* *e *ey* .
```

You can also extract erased files. E.g. if you want to extract the ’keynes’ program together with its documentation, you have to use one of these commands:

```
cp -u keynes &eynsdoc .
cp -u *ey* .
```

The ’-u’ means you want BDDE to salvage erased files, like ’keynsdoc’. Since the BetaDisk discards the first character of an erased files, BDDE substitutes it with an ampersand. By the way, all this works with ’ls’ as well.

1.15 Just another ‘od‘..

Let’s say you issue a simple ‘ls‘ and all you get is a lot of error-messages. So you want to see what the heck is wrong with track 0. Or you want to see what is on the Tasword II file on track 4 before copying it to disk. A simple file-viewer won’t get you very far since Anadisk added sector-ID’s. And when you’re using an older version of Anadisk even the sectors may not be in the correct order. In these cases ‘od‘ will save you without resorting to your old BetaDisk. Let’s see what’s on track 0:

```
od
```

‘od‘ will generate this output:

```
Track 0 (00h), sector 0 (00h)

  0 (00h)     63 6f 70 79 - 20 20 20 20 - 42 28 01 20 - 01 02 00 01
 16 (10h)     62 61 63 6b - 75 70 20 20 - 42 2c 01 24 - 01 02 02 01
 32 (20h)     73 63 6f 70 - 79 20 20 20 - 43 00 80 00 - 80 0d 04 01
 48 (30h)     66 6f 72 6d - 61 74 20 20 - 43 00 a0 00 - a0 0a 01 02
 64 (40h)     66 6f 72 6d - 61 74 2a 20 - 43 00 a0 00 - a0 0a 0b 02
 80 (50h)     73 63 6f 70 - 79 20 20 20 - 42 2a 01 22 - 01 02 05 03
 96 (60h)     62 61 63 6b - 75 70 20 20 - 43 00 80 00 - 80 08 07 03
112 (70h)     63 6f 70 79 - 20 20 20 20 - 43 00 80 00 - 80 0e 0f 03
128 (80h)     00 6f 70 79 - 20 20 20 20 - 43 00 80 00 - 80 00 0d 04
144 (90h)     00 6f 70 79 - 20 20 20 20 - 43 00 80 00 - 80 00 0d 04
160 (A0h)     00 00 00 00 - 00 00 00 00 - 00 00 00 00 - 00 00 00 00
176 (B0h)     00 00 00 00 - 00 00 00 00 - 00 00 00 00 - 00 00 00 00
192 (C0h)     00 00 00 00 - 00 00 00 00 - 00 00 00 00 - 00 00 00 00
208 (D0h)     00 00 00 00 - 00 00 00 00 - 00 00 00 00 - 00 00 00 00
224 (E0h)     00 00 00 00 - 00 00 00 00 - 00 00 00 00 - 00 00 00 00
240 (F0h)     00 00 00 00 - 00 00 00 00 - 00 00 00 00 - 00 00 00 00

Track 0 (00h), sector 1 (01h)

....
```

Sorry I didn't include all 16 screens.. It may not be very helpful either. That's why BDDE has its '-c' option (just like good old Unix). It shows the same information, but in ASCII.

```
Track 0 (00h), sector 0 (00h)

  0 (00h)     copy    B(. ....backup  B,.$....scopy   C.......format  C.......
 64 (40h)     format* C.......scopy   B*."....backup  C.......copy    C.......
128 (80h)     .opy    C........opy    C.....................................
192 (C0h)     ............................................................

Track 0 (00h), sector 1 (01h)

....
```

If there are any hackers reading: you know what to do with this kind of stuff, don't you? If you don't have any use for it just forget it. When using the ASCII-display every non-printable character is shown as a dot. If you really need to distinguish between dots and non-printable characters use the hexadecimal display.

Finally, you can select any track on the BetaDisk (dump). Just use the '-tx' option. Replace the 'x' with any valid logical track number, e.g. to dump the fifth track issue this command:

```
od -t4
```

If you select a track beyond the disk an error message is displayed.

1.16 SmartSeek

BDDE has two ways to access a track. The first is sequential, so first of all track 0 is read, then track 1 and so on. When using 'od' you could be in for a long wait. Let's say you want to dump track 150. If BDDE would access this track sequential 150 tracks of 4 kB would have to be read.

It can even jump back a few tracks when necessary but the chance of finding yourself in that kind of situation is very remote. Usually all files are sorted in order of allocation. BDDEs 'ls' doesn't sort anything..

When SmartSeek is active the dump is really acting like it's a disk. BDDE can randomly access any 'track'. However, when you never issue an 'od' or use 'cp' conditionally you won't need SmartSeek very often. SmartSeek is activated when you use the '-S' option. It actually works on every command, but since 'ls', 'df' and 'di' only access track 0 you won't experience any dramatic increases in performance when issuing these commands. An example of a sensible use of SmartSeek:

```
od -t150 -S
```

1.17 'dc': does not compute

BDDEs 'dc' is a tiny implementation of the Unix 'dc', but it has its own virtues. First, it is very well suited for programmers. Second, it has a lot of features, like a stack of 32 elements, a constant register and a host of commands, both stack-, mathematical and binary. The 'dc' program can resolve an integer expression within the range of the 'int' datatype of the supporting compiler, typically between -2^15 and +2^15 or -2^31 and -2^31. When the program detects an error it returns INT_MIN, typically -2^15 or -2^31.

To use 'dc' you must understand Reverse Polish Notation. This is a way to write arithmetic expressions. The form is a bit tricky for people to understand, since it is geared towards making it easy for the computer to perform calculations; however, most people can get used to the notation with a bit of practice. Reverse Polish Notation stores values in a stack. A stack of values is just like a stack of books: one value is placed on top of another. When you want to perform a calculation, the calculation uses the top numbers on the stack. For example, here's a typical addition operation:

```
1 2 +
```

When 'dc' reads a number, it just puts the value onto the stack. Thus 1 goes on the stack, then 2 goes on the stack. When you put a value onto the stack, we say that you push it onto the stack. When 'dc' reads the operator '+', it takes the top two values off the stack, adds them, then pushes the result back onto the stack. This means that the stack contains:

```
3
```

after the above addition. As another example, consider:

```
2 3 4 + *
```

(The '*' stands for multiplication.) 'dc' begins by pushing the three numbers onto the stack. When it finds the '+', it takes the top two numbers off the stack and adds them. (Taking a value off the stack is called popping the stack.) 'dc' then pushes the result of the addition back onto the stack in place of the two numbers. Thus the stack contains:

```
2 7
```

When 'dc' finds the '*' operator, it again pops the top two values off the stack. It multiplies them, then pushes the result back onto the stack, leaving:

```
14
```

The following list gives a few more examples of Reverse Polish expressions. After each, we show the contents of the stack, in parentheses.

```
7 2 -           (5)
2 7 -          (-5)
12 3 /          (4)
-12 3 /        (-4)
4 5 + 2 *      (18)
4 5 2 + *      (28)
4 5 2 * -      (-6)
```

The internal constant register (which may be accessed by the '!' and '@' commands) is assigned to the result of the previous calculation. So after this line:

```
2 4 +
```

The internal constant register holds '6'. If you want to preload it you can type:

```
6 dup !
```

Or even:

```
6
```

And it you want to retrieve it for your next calculation, just type use '@', like:

```
@ 4 %
```

However, you can also load it during the calculation, e.g. this does a ’ROT’ or ’rotate’ instruction (for those of you who know Forth):

```
1 2 3 ! swap @ swap
```

Finally, you probably want to know how to exit ‘dc‘. Very easy, just type "quit" or just "q". Remember, "quit" must be issued on an empty line. In the next section we’ll present all the commands ‘dc‘ has to offer.

1.18 ‘dc‘ commands

In this section ’x’ and ’y’ represent numbers on the stack. The rightmost value is on top of the stack. These operators are available:

1.18.1 Stack operators

x y s	(swaps x and y on the stack)
x d	(duplicates x on the stack)
x k	(drops x from the stack)
x y o	(duplicates x on the stack)
x !	(gets x from stack and stores it in memory)
@	(puts memory on stack)

1.18.2 Mathematical operators

x y +	(adds x and y and puts the result on the stack)
x y -	(subtracts x and y and puts the result on the stack)
x y *	(multiplies x and y and puts the result on the stack)
x y /	(divides x and y and puts the result on the stack)
x y %	(divides x and y and puts the remainder on the stack)
x y l	(takes x and y from stack and returns the smallest on the stack)
x y m	(takes x and y from stack and returns the largest on the stack)
x a	(takes the absolute of x and returns it on the stack)
x n	(negates x and puts the result on the stack)

1.18.3 Binary operators

x y | (ORs x and y and puts the result on the stack)

x y & (ANDs x and y and puts the result on the stack)

x y ^ (XORs x and y and puts the result on the stack)

x y > (shifts x right y and puts the result on the stack)

x y < (shifts x left y and puts the result on the stack)

x ~ (NOTs x and puts the result on the stack)

1.18.4 Writing commands in full

The s, d, k, o, l, m, a and n commands may be written in full:

s => swap

d => dup

k => kill

o => over

l => less

m => more

a => abs

n => neg

1.19 ‘sh‘: an empty shell is better than half an egg

The most exciting new feature of BDDE is undoubtedly the shell. We’ve already used it in the previous sections and in my view it is a vast improvement over the old commandline-type BDDE.

In this section we’ll prove *how* useful. As a matter of fact, you’ll never have to leave the shell again to get to your Operating System or Spectrum Emulator. Even BDDEs famous Online Help has now turned into a *real* Online Help.

But how do you get to grips with this new user-interface? A nice place to start is the ‘man‘ command. If you are still looking at the ’$’ prompt, type:

```
man
```

BDDE will present you something like this:

```
ls      [options] [filelist]                  List files
cp      [options] <filelist> <directory>      Copy files
df      [options]                             Show free diskspace
di      [options]                             Show disk characteristics
od      [options]                             Dump a track
dc      [options]                             Desk Calculator
mount   [options]                             Mount a BetaDisk
!       [command]                             Execute external command
echo    [string]                              Show string
```

```
#       [string]                        Comment line
sleep   <seconds>                       Pause execution
cd      <directory>                     Change directory
exit                                    Exit shell
zx                                      Start Spectrum Emulator
guide                                   Start BDDE User Guide
help                                    Show standard BDDE helpscreen
man                                     Show this screen
env                                     Show BDDE variables
pwd                                     Show present working directory
```

We certainly know the first six commands. They are in fact the very core of BDDE and if you want to know more about them, start reading backwards! We’ve also seen ‘mount‘. We used that command to mount our dump-file. However, it can do more.

First of all, you can use ‘mount‘ to set options without actually doing anything. For instance these two commands are virtually equivalent:

```
mount -Fd:\temp\bdde.dmp -M0
ls -Fd:\temp\bdde.dmp -M0
```

Both commands will mount that file. The difference is, that ‘ls‘ will show you a list of filenames and ‘mount‘ won’t. You don’t even have to mount a file when you use ‘mount‘. This will set the debug-level to ERROR:

```
mount -D4
```

Nothing has changed, except that the debug-level has been modified. Well, nothing? Yes, it has done *something.* If you set options with ‘mount‘ the "FixedInterLeave"-flag is reset. We’ll discuss it in more detail later, but is used to freeze the interleave on an Anadisk-dump if the sector-information has been damaged. Nobody ever reported such an error, but if you encounter it you have to know that by using ‘mount‘ you affect this flag.

You can also issue ‘mount‘ without any options. By the way, this does *not* affect the "FixedInterLeave"-flag. BDDE will then report what data you have entered:

BetaDisk ’c:\temp\bdde.dmp’ mounted, interleave 1:1, raw dump

This doesn’t mean that you can actually access the dump-file; it just reflects the mount-settings. Note, that the interleave can change when you’ve actually accessed an Anadisk-dump. That’s the beauty of the Anadisk/BDDE combination: no more interleave worries.

We’ve also encountered ‘pwd‘. It works just like the Unix ‘pwd‘: it shows you the present working directory. It’s equivalent with this MS-DOS command:

```
cd
```

Its use is of course very limited. Combined with our ‘cd‘ command it works like a charm. BDDEs ‘cd‘ works like a Unix ‘cd‘: it makes the directory you enter the present working directory, even when it’s located on another drive. This example will show you the difference with a MS-DOS ‘cd‘:

```
$ pwd
c:\temp
$ cd d:\4th
$ pwd
d:\4th
$
```

You see? When you've issued a MS-DOS 'cd' you still have to change drives; with a Unix or BDDE 'cd' you don't.

Okay, now you're there. Can you also see what files there are in that directory? Yes and no. BDDE has no built-in command to do that. But you can access your Operating System shell without leaving BDDE. Try this:

```
! ls -x
```

Or when you're working with MS-DOS:

```
! dir /w
```

Don't forget the space! If you do BDDE will issue an error-message. The '¡ command is a quick and easy way to issue a shell-command. You can format a diskette, copy files, even run your wordprocessor, without ever leaving BDDE. Sorry, this command is *NOT* available on the MS-Windows version.

The 'echo' command will simply display a string. Let's examine this dialogue:

```
$ echo This is BDDE
This is BDDE
$
```

Why the heck did I include this one? Well, you can use it to run scripts. The demo of BDDE V1.7e used this command to display quite a lot of information. We'll tell you all about scripts. But not now.

The '#' command seems even more useless. It does nothing, absolutely nothing at all. You can use it to comment scripts, but in an interactive environment it is less than useless.

Then there is the 'sleep' command. If you enter:

```
sleep 10
```

BDDE will wait 10 seconds before returning the '$'-prompt. Very useful when writing a demo of your new software. That's why it's included ;-). Not available on MS-Windows either.

'exit' does what 'exit' should do: it leaves the shell and returns you to the Operating System. It works just like the Unix or MS-DOS version of 'exit'.

'help' simply shows you the standard BDDE helpscreen. This can be very useful if you forgot which options are available or how to use them. Note that it will *NOT* throw you out of the shell and back to the Operating System.

'env' will be discussed in the Technical Manual.

The 'zx' command will start your Spectrum Emulator in the MS-DOS version. Yes, you're only three keystrokes away from your emulator. And when you're done you return to the BDDE prompt. Isn't that neat?

You can for instance extract all the files of a game, start up the emulator, modify the BASIC-loader, load the game, make a snapshot (if supported by your emulator) and return to the BDDE prompt, ready for the next game!

But how does BDDE know what emulator you got? Well, you told him by using the '-Ex' option. I suggest you include it in your BDDE.INI file or BDDEINIT variable. Simply substitute the 'x' with the command you normally use to start your emulator. It doesn't matter whether you call it with a batch-file, a 4DOS-alias or an Operating System command. E.g. when you have a batch-file in your path called SPECTRUM.BAT you can enter:

```
-Ec:\batch\spectrum.bat
-Espectrum.bat
-Espectrum
```

They all work. Finally, there is the ‘guide‘ command. This can be very useful when ‘man‘ and ‘help‘ do not provide enough information to help you with a particular problem. It starts BDDE-MAN.EXE, which should be included in this package. Just place it in the very same directory where BDDE is located. If you put it in another directory ‘guide‘ won’t work.

Well, I think we’ve covered everything. Except for some odd technical stuff. First, these commands can only be entered from the shell. They cannot be entered from the Operating System shell, e.g.

```
bdde mount -Fd:\temp\bdde.dmp
```

won’t work! Second, they do *not* access the dump. Only the core-commands (‘ls‘, ‘df‘, ‘di‘, ‘cp‘, ‘od‘) do. Third, the shell is *NOT* re-entrant. You can issue:

```
$ sh
```

but it won’t work. If you want to change the prompt, you can with the ’-px’ option. The ’x’ stands for the prompt you want to see, e.g. if you want to play root:

```
p#
```

or mimic MS-DOS:

```
-pC>
```

or prefer a ‘ftp‘-like prompt:

```
-pbdde>
```

or want to disable the prompt altogether (handy when you’re running a script):

```
-p
```

All that is possible, but remember: the prompt is fixed. You can’t make it reflect your current working directory. Yes, you guessed it: just like the Unix ‘sh‘ ;-).

1.20 Running BDDE from the prompt

May be you want to use BDDE within a shell-script or batch-file. May be you want to print or redirect its output. May be you don’t like shells. Well, you can run BDDE from you Unix- or MS-DOS prompt.

That is, most commands. You cannot use any of the shell-commands we presented in the section above, like ‘mount‘ or ‘exit‘. That wouldn’t be very useful either. But ‘di‘, ‘df‘, ‘dc‘, ‘cp‘, ‘ls‘, ‘od‘ and ‘sh‘ are available. Try:

```
bdde ls -Fbdde.dmp -M0 -i1
```

Yes, it works! Just like in the shell. You only have to add ‘bdde‘ before the actual command. Please note, that when you’re working with a Unix-shell like ‘ksh‘ or ‘sh‘, some characters have to be protected from the shell. This won’t work:

```
bdde ls -Fbdde.dmp -M0 -i1 music*
```

However, this will:

```
bdde ls -Fbdde.dmp -M0 -i1 "music*"
```

When you’re working with MS-DOS you won’t run into this problem, since the MS-DOS shell doesn’t expand filenames at the prompt. However, if the filename contains characters that have a special meaning to your Operating System like spaces or tabs, you’ll have to enclose them by quotes too, like:

```
bdde cp "bt,ks&ei" "I wish.." .
```

You can also run scripts from the prompt, like BDDE.BSH which we included. Running scripts is very easy:

```
bdde sh < bdde.bsh
```

In fact, we redirect standard input to the file BDDE.BSH. You can do anything in a script that you can do from the shell. If there is an error BDDE will report it. However, *never* forget to terminate a script with ‘exit‘! If you don’t BDDEs behavior is undefined. Most likely your PC will hang.

You can also redirect standard output and standard error as we will see in the following section.

1.21 Piping and redirecting

I think you MS-DOS junkies have found out by now that I like to use another Operating System. Right. It’s Unix. Unfriendly, chaotic, but awesome (believe me). If you compare the script language of ‘ksh‘ or ‘csh‘ to the batch-language of MS-DOS it looks like the latter has been designed a very drunk COBOL-programmer. Every Unix command has a very small but clearly defined task. The only way you can get a Unix-box to work is by putting these commands together by pipes, redirections, environment variables, etc. And you can tune the operation of these commands by using options.

This philosophy has effected BDDE. There is no paging option, no printing option, or any option that every MS-DOS programmer usually includes in his program. But you can do all this with standard MS-DOS or Unix tools:

Print ‘ls‘ output:

```
bdde ls -l -Fbdde.dmp -M0 -i1 > prn
bdde ls -l -Fbdde.dmp -M0 -i1 | lp (Unix)
```

Show ‘ls‘ output per page:

```
bdde ls -l -Fbdde.dmp -M0 -i1 | more
bdde ls -l -Fbdde.dmp -M0 -i1 | pg (Unix)
```

Remove headers from ‘od‘ output:

```
bdde od -c -Fbdde.dmp -M0 -i1 | find h) | find /v sector
bdde od -c -Fbdde.dmp -M0 -i1 | grep h\) | grep -v sector (Unix)
```

Print 'od' output without headers:

```
bdde od -c -Fbdde.dmp -M0 -i1 | find h) | find /v sector > prn
bdde od -c -Fbdde.dmp -M0 -i1 | grep h\) | grep -v sector | pr | lp
```

Most MS-DOS users never utilize redirection and piping. That is a shame, because it is *very* powerful and has a great potential. Many examples in this documentation have been generated by redirecting the output of BDDE, e.g.:

```
bdde ls -l -Fbdde.dmp -M0 -i1 > bdde.txt
```

Of course, there a very few tools available when you use standard MS-DOS, and there are even fewer programmers that support this feature, although it is very easy (when you program in C) and even portable! Standard MS-DOS provides only 'sort', 'more' and 'find', which are in fact (less capable) equivalents of the Unix commands 'sort', 'more' and 'grep'. But even then you can do quite a lot..

By now you must have blisters on your fingers from typing the same options over and over again. We've got another way to do that, as we will explain in the following section.

1.22 Initializing BDDE

You can initialize BDDE in various ways. If you think the only thing that is coming is the BDDEINIT variable you're wrong. BDDE will collect its settings from various sources. First of all there is the initialization-file. If you're working with Unix, it's called '.bdderc' and ought to be placed in your $HOME-directory. If you're working with MS-DOS it's called BDDE.INI and is placed in the same directory where BDDE.COM is. If you do not want to place it there, you can place it somewhere else. Just add this line to your AUTOEXEC.BAT:

```
set BDDE=<directory>
```

e.g.

```
set BDDE=d:\spec
```

Or if you're working with Unix, add a line like this to your .profile:

```
BDDE=/users/habe/data; export BDDE
```

What does such an ini-file look like? Well, just like the commandline. However, all the arguments have their own line. BDDE can not generate such a file on its own. You have to create it with an ASCII-editor like 'vi', EDLIN, etc. This is an example of an ini-file:

```
-Fc:\spec\bdde.dmp
-M0
-i3
-C4
-S
```

When BDDE is started it will look for this file first. If it isn't there it will continue without an error, since you don't have to have an ini-file. Then it will look for an environment variable called BDDEINIT. The format for BDDEINIT is a little different. You issue the arguments just like on the commandline. If you're working with MS-DOS, you set BDDEINIT like this:

```
set BDDEINIT=-Fc:\spec\bdde.dmp -M0 -i3 -C4 -S
```

If you're working with Unix use this command:

```
BDDEINIT="-F/home/habe/bdde.dmp -M0 -i3 -C4 -S"
export BDDEINIT;
```

And, just like we said before, BDDEINIT is interpreted *after* the ini-file. Then the command-options are interpreted. However, you can also add an ini-file on the commandline. Its format is exactly the same as BDDE.INI or '.bdderc'. Just use the '-Ix' option:

```
bdde ls -l -Ialien.ini
```

This way you can keep the options needed for a certain task neatly in an ini-file so you don't have to find it all out again or issue all the options at the commandline. You don't have to erase ini-files or environment-variables, since BDDE will only use the last value entered and cancel out all previous entered values. E.g. if your ini-file contains:

```
-Fc:\spec\bdde.dmp
-M0
-i3
-C4
-S
```

and you issue this command:

```
bdde cp -Fd:\temp\games1.dmp -M1 -i1 -C3 -S- * .
```

BDDE will still use file D:\TEMP\GAMES1.DMP, treat it like an Anadisk-dump with interleave 1:1, disable SmartSeek and produce files for the SpecEm emulator. Most single character-options can be turned off by appending a dash.

1.23 Using listfiles

Working from the commandline has another advantage. Let's say you want to extract several multi-file games. In that case the list can become very long. So BDDE has another trick up its sleeve to make your life easier.

It works quite simple. Just issue this command:

```
bdde ls > filelist.txt
```

The file 'filelist.txt' will now contain this information:

```
index
FRED
keynes
bt,ks&ei
probe
musicflr
musicode
flagrec
HanSoft
C
I wish..
birthday
slide
<0>
```

Now fire up your favorite editor and edit the file. Just delete the files you do *NOT* want to extract, e.g. let's say we only want the index-program (including its data) and the 'musicfiler' program:

```
index
FRED
musicflr
musicode
<0>
```

You can now extract these files by issuing this command:

```
bdde cp @filelist.txt .
```

That's it! Easy as pie, ain't it?

1.24 I love it when a plan comes together..

Actually I'm doing somebody elses job down here. It should all be in the manuals of the Spectrum Emulator, but when you hate to switch manuals as much as I do continue reading.

Just a little tip before you get too enthusiastic and try to load in a multiple file program or something like that. Start off with something very easy, like a BASIC program. Be aware that BASIC programs which contain machine-code (e.g. in a REM statement) might *not* work! Because the memory layout of every emulator is different from that of a standard BetaDisk machine addresses may have been moved and this could easily "crash" the emulated Spectrum. However, BDDE comes with a small program that tries to emulate this layout while preserving all functionality so you don't have to hack a lot to get these programs up and running..

Sorry folks, I have to split you up again! First of all Gerton Lunters Spectrum Emulator for MS-DOS:

1.24.1 Z80

If you want to read a file into this emulator there are two possible angles you can take. First of all, you can assign the file to the RS232 port using a command-line option:

Z80 -u[SpecFile]

Of course [SpecFile] is a '.bas', 'chr' or '.nmb' file generated by BDDE. Another way is to enter the Emulator, press <F10> and choose <I>nput RS232. Then you can select your file using <D>isk. Leave the menu by pressing <ESC> and issue an RS232 command in Sinclair BASIC:

LOAD *"b"	to read a BASIC program
LOAD *"b" CODE <address>	to read a CODE program
LOAD *"b" DATA [var]<$>()	to read a DATA file

E.g.:

```
LOAD *"b" CODE 25000
LOAD *"b" DATA i$()
LOAD *"b" DATA i()
```

The border will flash for a while and hurray! There it is! You can now continue with section 1.25 unless you are using a copy of Gerton Lunters Z80 emulator V2.x.

If you want to read a file into this version of the emulator there are two possible angles you can take. First, if you use multiple .TAP file mode let BDDE write its files to your multiple .TAP file directory and start your emulator like this:

```
Z80 -td [directory_multiple_tap]
```

That is if you haven't added that option (yet) to your Z80.INI. You can also string .TAP files together and thus create a file you can use in single .TAP file mode.

To read a file, just issue any valid Sinclair BASIC tape-command:

LOAD "<name>"	to read a BASIC program
LOAD "<name>" CODE <address>	to read a CODE program
LOAD "<name>" DATA [var]<$>()	to read a DATA file

E.g.:

```
LOAD "format" CODE 25000
LOAD ""
LOAD "backup"
LOAD "mfiler" DATA i$()
LOAD "smo" DATA i()
```

There it is! It was worth the wait, wasn't it? Now continue with section 1.25.

1.24.2 Atari

It seems that Christian Gandler doesn't like to write long manuals. But it is very easy anyway. The file INHALT.INF contains all information about the Spectrum tape-files in the directory. BDDE does *not* create this file, since the emulator creates it anyway when you issue a LOAD statement. This may take a while (I wouldn't know since I don't have an Atari-ST), especially when you put a full 640 kB BDDE-floppy in your drive. Don't write-protect the disk! It is very hard for any program to write a file to disk when you have enabled write-protection..

To read a file, just issue any valid Sinclair BASIC tape-command:

LOAD "<name>"	to read a BASIC program
LOAD "<name>" CODE <address>	to read a CODE program
LOAD "<name>" DATA [var]<$>()	to read a DATA file

E.g.:

```
LOAD "format" CODE 25000
LOAD ""
LOAD "backup"
LOAD "mfiler" DATA i$()
LOAD "smo" DATA i()
```

A window will appear when you issue a null name (e.g. LOAD ""). At the bottom of the window you can select a directory. Other useful keys are:

<Up> and <Down>	Select a file
<Return>	Confirms your choice
<F1>	Reread INHALT.INF
<F2>	Regenerate INHALT.INF
<ESC>	"Tape loading error"

The name you'll have to use is exactly the same as your filename, e.g. when the Atari filename is "termina0", you'll have to load it into the emulator by issuing:

```
LOAD "termina0"
```

There it is! Now you can continue and read section 1.25 while I tribute the famous last words of this section to all SpecEm users..

1.24.3 SpecEM

To read a file, just issue any valid Sinclair BASIC tape-command. Just be sure that the files that you want to read in are in the same directory as your SpecEm emulator:

LOAD "<name>"	to read a BASIC program
LOAD "<name>" CODE <address>	to read a CODE program
LOAD "<name>" DATA [var]<$>()	to read a DATA file

E.g.:

```
LOAD "format" CODE 25000
LOAD "backup"
LOAD "mfiler" DATA i$()
LOAD "smo" DATA i()
```

The name you'll have to use depends on your filename. Just ignore the extension and trailing underscores, e.g. when the MS-DOS filename is "0ape_rd_.__b", you'll have to load it into the emulator by issuing:

```
LOAD "0ape_rd"
```

1.25 Mum, it doesn't work!

We really tried to test every possible BetaDisk we could lay our hands on, verify our documentation using different sources, etc. We even use the program ourselves and we never ran into trouble. However, every program has bugs and so has BDDE. Otherwise we would never find or fix any and we do. So you might experience problems.

But we'll give you support and a nice smile when you stop by for a beer. If you have any ideas for enhancement or help me to port this program to other Operating Systems or environments, please send an email to **hansoft@bigfoot.com**. This includes adding support for more Spectrum Emulators (I never realized how many there are!). Of course, we only support BDDE.

But what do you have to do when BDDE *really* doesn't work? It would be very nice if you send me your copy of BDDE and the dump that BDDE failed to convert. Then we can try to reproduce your problem. But if you try to convert your dairy in Tasword II and do not want so send me such personal or sensitive material please send me a diskette with your copy of BDDE and a debug listing. You can make a debug listing with this command:

```
bdde [command] <options> -D0 [filespec] [directory] > [filename]
```

But you should be able to think of that yourself after our nice piping-and-redirecting tutorial ;-).

However, you can diagnose or even fix most problems yourself. BDDE is loaded with features to allow you just that. We will present a few of them in the following sections.

1.26 Debug Level

A special feature of BDDE is the debug level. You can set the required debuglevel by issuing a '-Dx' option. The 'x' represents a number between 2 and 5. BDDE has 4 debug-levels. Initially used by the author to see if this beautiful program worked as intended, but now for your convenience:

(5) *(FATAL)*, the program aborts. BDDE can't recover from an error like this. Like there is no BetaDisk dump to read.

(4) *(ERROR)*. There is probably something very wrong, but the program can recover from the error. E.g. a wrong sector has been read (that's why we need the sector-IDs). If you make debug-level 4 the standard level, the program aborts also when it encounters a level 4 error. This is not always necessary. Sometimes you can safely continue.

(3) *(WARN)*. There is something of special interest going on. The program can easily handle this kind of situation. E.g. an erased file is skipped.

(2) *(INFO)*. The user just wants to know what is going on. E.g. a BetaDisk file has been converted to a MS-DOS file. This is the default.

1.27 Options that can make your day

-s When a different number of sectors has been allocated to a file then necessary (e.g. 2 sectors of 256 bytes have been allocated to a file of 256 bytes) this option comes in. It disregards the length of the file and forces BDDE to use the number-of-sectors entry. Use only when BDDE fails to produce a usable file.

-dx When a DATA file is saved the BetaDisk doesn't seem to be store any information about the variable-name. Since it's vital to store some value in the header BDDE picks a variable-name, starting with 'A'. Then you want to use another character to start with use this option, e.g. -dd, which starts off with 'D'. After assigning a character to a DATA file this value is incremented, so the next DATA file would get 'E' assigned to it. However, the Spectrum doesn't seem to care to which variable it was originally assigned as long as the type and size are correct. Use only when the Spectrum fails to read the DATA file.

-f This option fixes a variable-name. Usually used in combination with option -dx, e.g. 'bdde cp -dh -f ...'. This will assign variable 'H' to every DATA file on the dump.

-# The BetaDisk doesn't just discard the variable-name of DATA files, it even doesn't store what kind of file has been saved! This information is even more vital to the Spectrum. BDDE uses a smart algorithm to find out what kind of DATA file has been saved. It works excellent when using undamaged DATA files with no more than 127 dimensions. However, when for some reason this algorithm doesn't work you can force numeric arrays by using this option. Be careful: *all* arrays will be saved as numeric arrays when using this option.

-$ The same as above. Forces string arrays.

Note that when you're using these options in the shell, they are reset to their default-values after each call. Just like the '-l', '-c' and '-u' options. They are tools to fix certain errors, not standard features.

1.28 The interleave

This is a difficult subject and most users don't want to know about it and some don't have to know about it. Most users *never* have to do with the interleave, because BDDE is smart enough to sort it all out by itself.

However, it needs a little help. Anadisk is a very nice program and highly recommended. If you dump your disks using this program and include sector-IDs in the process you can forget all about interleaves. ADU can sort out the interleave as well, but BDDE can neither check nor correct the interleave.

Interleave errors are some of the most common errors when using BDDE. So in this section we will not only explain what to do about them, we will also provide you with a step-by-step strategy to solve them.

The interleave has to do with the way a disk is organized. Let's say you're working in some pizza parlor. The pizza is already cut in 8 slices and you have to put the slices in a box. Since this pizza parlor is highly automatized the pizza is lying on a turning table. However, there is a catch. Every slice is numbered and you'll have to put the slices in the box in the right order. So you wait until slice 1 appears. You pick it up and put it in the box as fast as you can. When you have turned back you see that you missed slices 2 and 3 and are facing slice 4. So you have to wait almost a full turn until you can pick up slice 2 again. That slows you down, doesn't it?

To prevent this you have to put slice 2 at position 4. So when you're ready to pick up a slice again, it's there! Of course you'll have to put slice 3 at position 7. And slice 4 at position 2. And so on.

Effectively, you've created an interleave of 3:1, while starting off with an interleave of 1:1. And that's the way *all* disks are organized, whether floppy or hard-disk. And just like you when packaging pizzas, they have an interleave that gives the highest performance. Well, that's all the explanation you boys are going to get. Now it's time for the Real Men.

BDDE supports all interleaves from 1:1 up to 8:1. I haven't got the faintest idea whether they are necessary or not. But if you need them, they're there. BDDE expects an interleave of 1:1 in accordance with most disk-analyzers. However, when you dump a disk with sector-IDs and BDDE reads a sector with a wrong ID, it will try to find out what the corresponding interleave is. If it is successful it will switch to that

interleave and continue without a glitch. Of course it is possible that BDDE switches the interleave more often, but only in rare cases of data-corruption.

When the going gets tough, the -i option gets going. Just add an interleave to -i, e.g. -i3. After that the interleave is fixed and BDDE won't try to change it again. Use with care! To determine the right interleave you'll have to follow this procedure:

First, try this command:

```
bdde di
```

If it produces something like this:

```
(ERROR) MountDisk: File 'bdde.dmp' probably not a BetaDisk; media descriptor: 00h
PHYSICAL DISK INFORMATION                LOGICAL DISK INFORMATION
-------------------------                ------------------------
Format....                   Unknown     Media descriptor.............   0h
Heads.........................  2        Total sectors.................. 2560
Cylinders.....................  80       Bytes per sector.............  256
Starting head.................  0        First sector of directory.....    0
Starting cylinder.............  0        Number of sectors on directory..  16
Starting sector...............  1        Maximum number of dir. entries..  128
Ending head...................  1        First sector of data area.....   16
Ending cylinder...............  79
Ending sector.................  16
```

then your disk-analyzer probably did not sort out the interleave properly. Now try:

```
bdde di -i2
```

It should produce something like this:

```
PHYSICAL DISK INFORMATION                LOGICAL DISK INFORMATION
-------------------------                ------------------------
Format....  40 tracks, single sided      Media descriptor.............  19h
Heads.........................  1        Total sectors.................  640
Cylinders.....................  40       Bytes per sector.............  256
Starting head.................  0        First sector of directory.....    0
Starting cylinder.............  0        Number of sectors on directory..  16
Starting sector...............  1        Maximum number of dir. entries..  128
Ending head...................  0        First sector of data area.....   16
Ending cylinder...............  39
Ending sector.................  16
```

If it still fails to produce a report like this try:

```
bdde di -i3
```

One of these should work in most cases. I've seen quite some BetaDisks, but none ever had an interleave of 4:1 or up. But you're welcome to try them. But we're not ready yet! Sometimes it just looks like BDDE has detected the right interleave, but the detection mechanism is not fail-safe. It is quite possible that e.g. '-i1' and '-i3' produce the same results. So extract a non-trivial BASIC-file (size 4 kB or up), e.g.

```
bdde cp -i1 keynes .
```

and read it into your Spectrum emulator. Now list it by issuing:

```
LIST
```

to your (emulated) Spectrum. If the listing looks fine, you've got the right interleave. If it doesn't, try the other one. Repeat the procedure for all possible interleaves until you got the right one. Then you can handle your dump as easily as the one we provided you with. But you *HAVE* to issue the corresponding '-ix' option with every command.

1.29 (Error)messages

BDDE can generate following messages. Usually only errors of debug-level 2 and higher will be reported, unless you specify another debug-level using the '-Dx' option.

(FATAL) FillBuf: Unexpected end of 'dumpfile'

(FATAL) FillBuf: Unknown error reading file 'dumpfile'

BDDE detected an End-Of-File while reading a dumpfile. You may have forgotten to dump the whole disk or specified the wrong mediatype with the '-Mx' option. Can also be caused by data-corruption in track 0.

(ERROR) GetTrk: Wrong track read: expected track x, got track y

(ERROR) CheckHeader: Invalid sector header: Fys. track (xxh)/Log. track (xxh)

(ERROR) CheckHeader: Invalid sector header: Fys. track (xxh)/Calc. track (xxh)

(ERROR) CheckHeader: Invalid sector header: Fys. side (xxh)/Calc. side (xxh)

(ERROR) CheckHeader: Invalid sector header: Fys. sector (xxh)/Calc. sector (xxh)

(ERROR) GetILeave: Invalid sector header: Fys. sector (xxh)/Calc. sector (xxh)

When you're using Anadisk and included sector-ID's BDDE incessantly checks which track/side/sector has been read. This error might be caused by inconsistency within the sector-ID itself. It might also be that the sector read wasn't the sector BDDE expected. Maybe the wrong file was read (e.g. COMMAND.COM). May be you didn't follow the instructions concerning Anadisk, specified the wrong mediatype or only dumped one side of a double-sided BetaDisk.

(ERROR) SeekTrk: Random access failed on track x of file 'dumpfile'

(ERROR) SeekTrk: Sequential access failed on track x of file 'dumpfile'

The first message indicates that SmartSeek failed. It should never occur. The second message indicates that a track before the current track was requested. This message should never occur when using a clean dump-file. It may indicate a rare case of data-corruption in the BetaDisk directory.

(FATAL) MountDisk: Unable to mount file 'dumpfile'

Mapping a physical disk to a directory is called 'mounting' in Unix. If BDDE fails to find the dumpfile this message will be issued. Usually due to typing errors.

(ERROR) MountDisk: Unable to assign buffer to file 'dumpfile'

(ERROR) CopyFile: Unable to assign buffer to file 'DOSfile'

BDDE has two fairly large buffers. The dump-file buffer is exactly the size of a physical track (including sector-IDs when necessary). The file-buffer is larger, more than 10 kB so even the largest files can be written to disk in only three write-actions. If a buffer cannot be assigned to a file, this message is display. Actually, it is rather hypothetical since the buffers are not allocated on the heap.

(ERROR) MountDisk: File 'dumpfile' probably not a BetaDisk; media descriptor: xxh

BDDE warns you that this dumpfile does not look like a BetaDisk. A valid BetaDisk has an indication of its file-format, called a media descriptor. However, the value of this media descriptor is out of range. This always indicates data corruption on track 0.

(ERROR) GetAType: Too many dimensions: file 'DOSfile' written as CODE

(ERROR) GetAType: Too many elements: file 'DOSfile' written as CODE

(ERROR) GetAType: Corrupt data: file 'DOSfile' written as CODE

(ERROR) GetAType: Unknown type: file 'DOSfile' written as CODE

These messages are shown when there is something wrong with a DATA file and you didn't use the '-#' or '-$' options. The first message indicates that there are more than 127 dimensions and that is beyond the specifications of the detection algorithm. When the second message occurs there is really something wrong. The algorithm detected more than 65536 elements and that's more than 64 kB. The third message is displayed when a checksum error is detected. The last message is shown when an entirely new data-type has been found. Most of the time these last three messages indicate data-corruption. The file will still be written, but with a CODE header.

(INFO) GetAType: 'Betafile' (xx bytes) contains numeric array A (x elem. in y dims.)

(INFO) GetAType: 'Betafile' (xx bytes) contains string array A (x elem. in y dims.)

(INFO) GetAType: 'Betafile' (xx bytes) contains invalid array A (x elem. in y dims.)

The first two messages indicate that the detection algorithm performed perfectly. The last message will occur only when other DATA file errors have been displayed. It might be helpful in finding the cause of the error.

(WARN) PatchName: Rename file 'XX.XXX' to 'DOSfile'

BDDE will try to synchronize the names in the tape-headers with the filenames of a specific emulator. However, once BDDE starts using random filenames it actually loses control. You are warned that this file can't be read into your emulator without renaming it first. Will only be shown when using a tape-header conversion like SpecEm.

(ERROR) WriteBlock: Unknown error writing RS232 header to file 'DOSfile'

(ERROR) WriteBlock: Unknown error writing tape header to file 'DOSfile'

(ERROR) WriteBlock: Unknown error writing checksum to file 'DOSfile'

(ERROR) WriteBlock: Unknown error writing data to file 'DOSfile'

An error occurred while BDDE tried to write data to a DOS file. Most of the time these messages do not come alone. May be your disk is full or write-protected.

(ERROR) CopyFile: Unable to open file 'DOSfile'

BDDE is unable to create a DOS file. May be you specified an invalid target-directory. May be there are already too many files in the root-directory.

(ERROR) CopyFile: Error closing file 'DOSfile'

BDDE was unable to close the DOS file correctly. May be you removed the floppy or your disk is full. This message may be accompanied by other write-errors.

(INFO) RunLine: Program 'Betafile' runs from line xxxx

(WARN) RunLine: Unable to determine autoRUN linenumber of program 'Betafile'

The method BDDE uses to determine the autoRUN linenumber works with 99.9% of all programs. In that case the first message is printed, the BetaDisk filename and the linenumber itself. When the method fails the second message is shown.

(WARN) MakeEntry: BetaDisk file 'Betafile' may be an unconvertible snapshot

BDDE warns you that it has found a CODE file with a length of 192 sectors. This is an strong indication for a BetaDisk+ snapshot. BDDE can convert it, but you might be unable to use it. If anyone has some technical information concerning snapshots, write us!

(ERROR) MakeEntry: Unknown type 'X' in file 'Betafile'; CODE assumed

BDDE found another file-type than BASIC, CODE or DATA. So it was written to disk as a CODE file. Might be data-corruption or a BetaDisk+ disk.

(ERROR) CheckLength: File 'Betafile' needs xx sectors but allocates yy

The BetaDisk file has allocated another number of sectors than its filesize indicates. If you're unable to load the file correctly try the '-s' option.

(WARN) CheckLength: Filesize adjusted: 'Betafile' now xxx bytes long

BDDE warns you, that the previous error has occurred and is corrected by the '-s' option.

(WARN) MakeDir: Erased file '?etafile' found: skipped

BDDE warns it found an erased file. Since the '-u' option is not specified, the file will be skipped.

(ERROR) MakeDir: BetaDisk reported more files than BDDE found (xx vs. xx)
(ERROR) MakeDir: BetaDisk reported less files than BDDE found (xx vs. xx)

BDDE found more or less files than specified by the BetaDisk. Could be data-corruption. However, BDDE will still copy all the files it has found.

(WARN) MakeCopy: BetaDisk file 'Betafile' skipped

BDDE warns that the BetaDisk file could not be copied and is skipped. It will continue with the next file. Will always be accompanied by other open- and/or write errors on the equivalent DOS-file.

(INFO) MakeCopy: BetaDisk file 'Betafile' copied to 'DOSfile'

BDDE informs you that it successfully copied a BetaDisk file to its DOS file equivalent.

(FATAL) MakeDump: Track xx does not exist; disk has only xx tracks

The requested track is not present on the BetaDisk since the poor thing has only xx tracks. A double-sided 80 tracks disk has 160 logical tracks numbered from 0 to 159. A single-sided 40 tracks disk has 40 logical tracks numbered 0 to 39. All other disks have 80 tracks (0-79). When using the '-t' option please specify a valid track number. Possibly data corruption.

(FATAL) MakeDump: Disk access failed on track xx of file 'dumpfile'

Will always be accompanied by seek-errors. See SeekTrk() errors.

(FATAL) GetListFile: List-file 'ListFile' not found

BDDE checks whether an list-file is there before it tries to open it. This is always caused by typing errors at the '@' character.

(FATAL) GetListFile: Error reading list-file 'ListFile'

This might be caused by a physical disk error or not enough memory. If it is a physical disk error you cannot access it with other utilities (e.g. copying it). Otherwise you might have consumed too much memory using init-files.

(WARN) GetOptions: Illegal option '-X' ignored

You specified an option that is not supported by BDDE. BDDE will ignore the option and continue.

(ERROR) GetIniOptions: Init-file 'InitFile' not found

BDDE checks whether an init-file is there before it tries to open it. This is always caused by typing errors at the '-I' option or init-files that are not properly configured. If you can't find the error check your init-file(s) and $BDDEINIT.

(ERROR) GetIniOptions: Error reading init-file 'InitFile'

This might be caused by a physical disk error or not enough memory. If it is a physical disk error you cannot access it with other utilities (e.g. copying it). You might have created a loop in your init-files (e.g. "bdde.ini" calling "atari.ini" and "z80v2.ini" calling "bdde.ini")!

(ERROR) GetAllOptions: $HOME not defined; searching '.bdderc' in current directory

Unix only. BDDE needs the $HOME variable to locate your .bdderc file. If it isn't set BDDE might not be able to find your default init-file. Since this might contain vital information about how your conversion has to be executed, it warns you.

(ERROR) GetAllOptions: Definition of '$BDDEINIT' too long; truncated

BDDEINIT may only be 256 bytes long. If it is longer BDDE cannot interpret it completely. This is to warn you that BDDE will truncate the contents of BDDEINIT. As a consequence some settings will be incomplete or even missing.

(ERROR) GetAllOptions: Definition of 'drive:\path\bdde.com' too long; truncated

DOS only. BDDE needs the full pathname of BDDE to locate your BDDE.INI file. If it gets truncated BDDE might not be able to find your default init-file. Since this might contain vital information about how your conversion has to be executed, it warns you.

(ERROR) GetAllOptions: Definition of 'bdde-man.exe' too long; truncated

DOS only. BDDE needs the full pathname of BDDE_MAN.EXE to locate your BDDE-MAN.EXE file. If it gets truncated BDDE might not be able to execute the 'guide' shell command.

(FATAL) GetFileList: Target directory 'directory' does not exist

BDDE first checks whether the target directory you specified exists. If it doesn't BDDE terminates. All files couldn't have been opened anyway.

(ERROR) UnMountDisk: Error unmounting file 'dumpfile'

BDDE could not close the dump-file correctly. Will probably be accompanied by other read-errors.

(ERROR) GetAllOptions: No memory on heap left to hold init-file directory

(FATAL) GetAllOptions: No memory on heap left to hold $BDDEINIT

(FATAL) ExecShell: No memory on heap left to hold 'mount' options

(FATAL) ExecCommand: No memory on heap left to hold internal call

Sometimes BDDE needs additional memory to perform certain operations. BDDE allocates this memory automatically and frees it as soon as possible. However, sometimes no memory is available. If the operation BDDE tries to perform is crucial BDDE aborts. If not, BDDE continues but its functionality is restricted. Although nobody ever reported such an error, you might run into a message like this. The only advise I can give you: don't make your filelists too long!

(INFO) ExecShell: Exiting shell; xx (fatal) error(s) encountered

While you were using the shell you entered the 'exit' command. BDDE terminates. It also report how many fatal errors it encountered during this sessions. The number of errors represents the number of times BDDE would have terminated if you had entered the command from the Operating System prompt.

(WARN) ExecShell: Illegal command 'xxxx' ignored

You have entered a command which is unknown to the shell. Remember that some Operating Systems allow more commands than others. In this version of BDDE the command may not be implemented. Check your 'man' command. Check this manual. Otherwise it is probably a typo.

1.30 Tips and tricks

1. The listings produced by 'ls' and 'cp' might be very useful when transferring complicated programs (e.g. games) to disk. Print them.
2. Convert all your BetaDisks to dump-files before using BDDE. They can be handled more easily that way. Compress them to self-extracting files, e.g. with ARJ. You can put three or more compressed dumps on a single 720 kB floppy.
3. If you try to dump a BetaDisk using Anadisk and errors occur, write the faulty tracks down somewhere. When you generate 'ls' output (First Tk/Sd/Sc) you can easily find out which files are still usable.
4. Even when BDDE doesn't support your emulator (yet) you might still be able to use it! How can I find this out? Easily. If your emulator has tape-support try the Atari-ST conversion or the .TAP files. If it has RS232 support, try the MS-DOS conversion. May be you only have to rename the file.

1.31 Other programs written by this author

True, they do generally not feature flashy user-interfaces, but they are small, powerful and run under Linux, Unix, DOS and MS-Windows, so you can use them virtually everywhere. Furthermore, we also provide you with the full source-code, so you can change or port the programs yourself. Please, note that all our programs are GPL-ed. You can get the most recent versions of these programs at **http://hansoft.come.to**.

TEMP The most powerful printer utility ever made. String- and/or character substitution, native spooler-support, intelligent page-breaks, partial printing, user-defined headers and footers, initialization with environment variables or files, support for multi-wildcards and pipes, etc. It can convert textfiles, do mail-merging, and much much more.

4TH Toolkit to add a script-language to your own program. Very small and very fast (typically 5 times slower than a full compiler) by using semi-compiled H-code. Based on Forth, so users can get plenty of material to learn the language. 4TH can even run Forth programs with minimal changes. Adding your own commands couldn't be simpler, even with a skeleton knowledge of C. With 4TH you can write a compiler in a few lines of C.

1.32 Finally..

- The MS-DOS Spectrum Emulator is a product of G.A. Lunter
- The Atari-ST Spectrum Emulator is a product of Christian Gandler
- ARJ is a product of Robert K. Jung
- AnaDisk is a product of Sydex
- ADU is a product of AME Computing Systems
- Norton Utilities is a product of Symantics
- Unix is a product of Novell
- DRC is a product of David Harris
- MS-DOS is a product of MicroSoft
- PC-NFS is a product of Sun Laboratories
- Tasword II was a product of Tasman Software
- ZX Spectrum was a product of Sinclair Research Ltd.
- BetaDisk was a product of Technology Research Ltd.

"All other trademarks mentioned in this documentation are property of their respective owners."

Many thanks to Marcel and especially Jack Verheydt for their patience and support while adding the Atari-conversion. Thank you, boys for believing in me and this program!

Thank you Tinus v/d Wouw for your BetaDisk+ snapshot. I still can't convert it, but now BDDE can at least issue a warning! And your BetaDisk with interleave 2:1 was very useful too!

Many thanks to all users of the 'Tatort BBS' who downloaded this program, especially sysop Dick Pluym and Johan Muizelaar. Another thank-you to Bernard Lutz who revived this program after *eight* years!

And last but not least Gerton Lunter, who really did write a great Spectrum Emulator!

The read-me program was produced by David's Readme Compiler (DRC).

Chapter 2

Frequently Asked Questions

I get the same questions over and over again. So I thought it was time to produce a handy 'FAQ' sheet. Although it is not very long, all the questions people asked to me to far are covered here. Your's might be there too!

Q: I tried to use the '-C4' option to produce .TAP files. However, Gertons Emulator rejects these files. How come?

A: You might be using an old shareware version of BDDE (version 1.7d or lower). If you want to convert BetaDisk files to Z80 V2.x you'll have to use the '-C1' option.

Q: I'm using Anadisk and it reports nothing, so I assume everything is going fine. However, when I try to convert the file all BDDE produces is a lot of sector-header errors ("Invalid sector header .."). I tried every possible interleave but nothing seems to work. Will registering get me there?

A: No! If you can't produce any usable stuff with the shareware version, you can't do it with the registered version either. That's not BDDE fault. BDDE can only work if your disk-analyzer produces valid dumpfiles. Anadisk (in most cases) seems to 'miss' the first sector and only dumps sector 2 and up. If you examine your dumpfile carefully, you will see it is smaller than it should be. Contact your disk-analyzer reseller.

The whole thing seems to be an unpleasant combination of hard- and software, so trying another computer or disk-analyzer might work. If you have your dumps, you won't need the services of a disk-analyzer anymore, since BDDE can now do the job. If you're really desperate I can do a full conversion for you, but it will cost you. Please contact me for details.

Q: BDDE only produces a lot of files, but when I load and try to run program 'X' it won't work. Can't you produce .Z80 files?

A: Not directly. I would take a lot of artificial intelligence to work out all the dependencies between the BASIC-loader and the CODE-files, and I wonder if it would ever work nicely. But it is not very difficult to get a program up and running. Most of the time it just takes a few changes in the BASIC-loader. An example:

```
              Addr   Len    B/D First    First Last      First
Filename Type Len(B) Run(C) NEG Tk/Sd/Sc Abs.S Abs.S #Sc Tk/Sc
============================================================
BLUP 1   CODE  16384   8224   0  25/0/03   802   829  28 50/02
BLUP 2   CODE  25000   8224   0  25/1/15   830   908  79 51/14
BLUP 3   CODE  45000   8224   0  28/0/14   909   989  81 56/13
BLUPPER! BASIC   270    254 242  30/1/15   990   991   2 61/14
```

This is an imaginary game on a BetaDisk. It consists of three CODE-blocks and a BASIC-loader. If you convert these files to .TAP files they're called "blup1.tap", "blup2".tap, "blup3.tap" and "blupper!.tap". Now we read "blupper!.tap".

```
1 REM ! STEP n!# STEP n STEP RESTORE a
2 CLEAR VAL "24999": BORDER NOT PI: INK NOT PI: PAPER NOT PI: CLS : LET n$="BLUP "
3 FOR i=SGN PI TO VAL "3"
4 RANDOMIZE USR VAL "15363": REM : LOAD n$+STR$ (i)CODE
5 NEXT i
6 RANDOMIZE USR (PEEK VAL "23300"+VAL "256"*PEEK VAL "23301")
12 RANDOMIZE USR a: REM : SAVE n$+"3"CODE VAL "45e3",VAL "20535"
13 RANDOMIZE USR (PEEK VAL "23635"+VAL "256"*PEEK VAL "23636"+VAL "5")
14 RANDOMIZE USR a: REM : SAVE n$ LINE SGN PI
15 GO TO VAL "6"
```

This is the typical code of an protection-cracking interface. Packed to make it small and thus virtually unreadable. The "SAVE" code is not very interesting. The interface generated that to SAVE the game to disk (if you think about it this makes perfectly sense). That code can be eliminated since it isn't needed anymore (we want to load the game):

```
1 REM ! STEP n!# STEP n STEP RESTORE a
2 CLEAR VAL "24999": BORDER NOT PI: INK NOT PI: PAPER NOT PI: CLS : LET n$="BLUP "
3 FOR i=SGN PI TO VAL "3"
4 RANDOMIZE USR VAL "15363": REM : LOAD n$+STR$ (i)CODE
5 NEXT i
6 RANDOMIZE USR (PEEK VAL "23300"+VAL "256"*PEEK VAL "23301")
```

The next thing is to eliminate the BetaDisk-code. We want to load the game from our (virtual) cassetterecorder. The call to "USR 15363" or "USR 15360" is significant. It's located in line 4 and we have to delete it up to the point where "LOAD" starts.

```
1 REM ! STEP n!# STEP n STEP RESTORE a
2 CLEAR VAL "24999": BORDER NOT PI: INK NOT PI: PAPER NOT PI: CLS : LET n$="BLUP "
3 FOR i=SGN PI TO VAL "3"
4 LOAD n$+STR$ (i)CODE
5 NEXT i
6 RANDOMIZE USR (PEEK VAL "23300"+VAL "256"*PEEK VAL "23301")
```

But we're not quite ready yet! The string n$ contains "BLUP ", so when it enters the loading loop it tries to load "BLUP 1.tap", "BLUP 2.tap" and "BLUP 3.tap". However, these are not valid MS-DOS filenames! Therefore a lot of code in BDDE has gone into converting these filenames into valid MS-DOS filenames. In other words, BDDE changes these filenames. To get the loader working, we have to:

1. Convert all characters to lowercase.
2. Erase all spaces.
3. Erase all the other characters that MS-DOS doesn't allow to be part of filenames, e.g. "<", "*", "|".

In this case it is very easy: remove the spaces and change "BLUP" to "blup":

```
1 REM ! STEP n!# STEP n STEP RESTORE a
2 CLEAR VAL "24999": BORDER NOT PI: INK NOT PI: PAPER NOT PI: CLS : LET n$="blup"
3 FOR i=SGN PI TO VAL "3"
4 LOAD n$+STR$ (i)CODE
5 NEXT i
6 RANDOMIZE USR (PEEK VAL "23300"+VAL "256"*PEEK VAL "23301")
```

Now check whether your setting on .TAP files points to the right directory and type:

```
RUN
```

The game should now load itself. You can now also convert it to an .Z80 file if you wish. Check the manual on your Spectrum Emulator.

Q: Can I use BDDE to convert datafiles to MS-DOS?

A: Yes, you can. Choose '-C0' conversion to extract them headerless. Further conversion depends on the dataformat and may require some editing or programming. Sometimes you can use the Spectrum Emulator to convert certain files by printing them to disk, e.g. by using the Z80 Emulator.

Q: Some programs convert just fine, but when I try to RUN them they crash. What happened?

A: Check if your BASIC-loader contains embedded machinecode. Some programs are loaded with that stuff! You can either correct it manually (only recommended when the machinecode-part is not too big) or register. The registered version of BDDE contains a program that corrects this problem in most cases.

Q: BDDE does not seem to work properly. What should I do?

A: You can either try to find/correct the problem yourself. That is quite possible because BDDE contains a lot of diagnostic code and comes with a Developers Guide, which shows you all the BDDE internals and how to use the built-in diagnostics. But, beware no bug has been reported (yet)!

Chapter 3

Developers Guide

3.1 Introduction

May be you are a C lover, just like me and want to experiment with the sourcecode of BDDE and its related functions or utilities. You can. Be my guest. That's why the source-code is included. BDDE is a moderately complex program. If you are a reasonably well experienced C-programmer you should have no trouble at all understanding and modifying it.

So for whatever reason you are bothering yourself to read this, be my guest. I cannot promise this document will still be there in the next release of BDDE (every document has to be updated and I hate it..) so enjoy yourself while it's there.

3.2 Functions

BDDE contains several sets of functions. They can be divided into four groups:

1. Internal functions
2. Low level access functions
3. Command functions
4. User interface functions

Most of these functions and their arguments can be shown during execution, except for BDDEs internal functions. It isn't even possible to list some internal functions, because this would result in eternal loops. We'll list all groups and their members here:

3.2.1 Internal functions

ErrMsg	The main engine for all error-handling. No significant message is issued outside this function. Also determines which action will be taken; continue or quit.
ShowSettings	Shows all significant variables and their values in specific debug situations.
PrintArgs	Shows two-dimensional character-arrays in specific debug situations.

3.2.2 Low level access functions

LogTS_to_AbsSc	Converts a logical tracknumber and a logical sectornumber to an absolute sector.
LogTk_to_FysTk	Converts a logical tracknumber to a physical tracknumber.
LogTk_to_FysSd	Converts a logical tracknumber to a physical side or: on which side is a logical track located.
FillBuf	Reads a number of bytes from disk (dumpfile) and puts them into a named buffer.
GetILeave	Tries to resolve a discrepancy between the actual sector-ID and the expected sector-ID by recalibrating the interleave. Works only with mediatype 1 ('-M1' option).
CheckHeader	Checks the integrity both internal and external of the sector-ID header. When these tests fail, it calls GetILeave(). Works only with mediatype 1 ('-M1' option).
GetTrk	Reads 16 sectors from the current position and puts them in the trackbuffer.
SeekTrk	Positions the virtual diskhead (filepointer) to the required 'track', then calls GetTrk() to read it in.
MountDisk	Mounts the disk (opens the dumpfile) and reads track 0.
UnMountDisk	Unmounts the disk (closes the dumpfile).
WriteBlock	Writes a number of bytes from a named buffer to the MS-DOS file.

3.2.3 Command functions

DiskInfo	Directly supports the ‘di‘ command.
ShoSpace	Formats the "xx kb, (yy sectors)" lines of VolInfo. Supports the ‘df‘ command.
VolInfo	Directly supports the ‘df‘ command.
PatchName	Patches a filename when a MS-DOS file with the same name already exists on disk. Supports the ‘cp‘ command.
GetName	Converts a BetaDisk filename to a MS-DOS filename according to the conversion-type. Supports the ‘cp‘ command.
MakeFlag	Constructs a block-header for the .TAP file conversion. Supports the ‘cp‘ command.
MakeRS232Hd	Constructs a RS232 header for Z80 .SAV file conversion. Supports the ‘cp‘ command.
MakeTapeHd	Constructs a tape header for Gandler, SpecEm and .TAP file conversions. Supports the ‘cp‘ command.
GetAType	Determines whether the DATA-file written contains a character- or a number-array. Supports the ‘cp‘ command.
GetFType	Converts the BetaDisk filetype (DATA, BASIC, CODE) to the Spectrum filetype (BASIC, CHAR, NUMBER, CODE). Supports the ‘cp‘ command.
RunLine	Determines the autoRUN linenumber of BASIC files. Supports the ‘cp‘ command.
FastTrk	Writes all tracks from a BetaDisk-file to an Emulator- file, except for the last one. Supports the ‘cp‘ command.

CopyFile	Copies a BetaDisk-file to an Emulator-file. Supports the ‘cp‘ command.
DirInfo	Writes information about a BetaDisk-file to screen. Supports the ‘ls‘ command.
CheckList	Checks whether the file about to be processed is in the filelist. The filelist also allows wildcards. Supports both the ‘ls‘ and ‘cp‘ command.
CheckLength	Checks whether the length-information is consistent and adjusts it if necessary. Support both the ‘ls‘ and ‘cp‘ command.
MakeEntry	Converts a BetaDisk directory-entry to a BDDE directory-entry. Supports both the ‘ls‘ and ‘cp‘ command.
MakeDir	Constructs the BDDE directory from the BetaDisk directory. Directly supports ‘ls‘. Also supports the ‘cp‘ command.
MakeCopy	Copies an entire BetaDisk to Emulator-files. Directly supports the ‘cp‘ command.
DumpHex	Dumps a sector in hex. Supports the ‘od‘ command.
DumpAscii	Dumps a sector in ASCII. Supports the ‘od‘ command.
DumpSect	Dumps a sector. Supports the ‘od‘ command.
MakeDump	Dumps a track. Directly supports the ‘od‘ command.
MakeCalc	Enters calculator-mode. Directly supports the ‘dc‘ command.

3.2.4 User interface functions

CleanUp	Cleans up all the mess after a call in BddeShell(), like freeing memory, resetting variables and unmounting the disk.
Help	Shows a helpscreen and exits.
GetListFile	Reads a listfile and puts it in an array of filenames.
GetIniOptions	Reads an initfile and puts it in an array of options.
GetOptions	Interprets an array of options and sets BDDE variables.
GetFileList	Gets the filelist, if any. Also checks whether a target-directory exists when executing a ‘cp‘ command.
GetAllOptions	Gets default-initfile, gets $BDDEINIT and gets commandline options.
Execute	Executes encoded command.
ExecMount	Executes the ‘mount‘ command when it is called from the shell.
ExecShell	Executes all shell-commands except ‘mount‘.
ExecCommand	Executes the "core"-commands when called from the shell. In fact it is an interface between BddeShell() and Execute().
BddeShell	The main execution-loop of the shell. Determines whether it is a shell-command or a core-command.
GetCommand	Encodes command from commandline.
main	Gets command and options. Mounts disk and executes command. Finally unmounts disk. Entrypoint for BDDE.

3.3 Variables

Most variables are maintained by BDDE itself. However, there are some you can set yourself using options. Here is a brief listing of these variables and the options that set them.

uchar O_odTrk

Ranges: Between 0 and 159

Option: -t

Default: 0

char O_DbgLvl

Values: 0 Debug messages

1 Program flow messages

2 General information messages

3 Warnings

4 Severe warnings

5 Fatal errors

Option: -D

Default: 2

char O_DataType

Values: 1 Number array

2 String array

3 Unknown array

Options: -$, -#

Default: 3

char O_ConverType

Values: 0 Headerless

1 Z80 Emulator (Lunter)

2 Atari Emulator (Gandler)

3 SpecEm (Phair)

4 Z80 V2.x Emulator (Lunter)

Option: -C

Default: 1

char O_MediaType

Values: 0 Dumpfile without sectorheaders

1 Dumpfile with Anadisk sectorheaders

Option: -M

Default: 1

char O_InterLeave

Ranges: between 0 and 7 (interleave - 1)

Option: -i

Default: 0

char O_DataChar

Ranges: between ’a’ and ’z’

Option: -d

Default: ’a’

char O_BetaDump

Contents: filename of the dumpfile

Option: -F

Default: ""

char O_Spectrum

Contents: command to call the emulator

Option: -E

Default: "Z80"

char O_Prompt

Contents: string to display as shell-prompt

Option: -p

Default: "$ "

These are boolean values, so they can either be true or false.

char _LNG_

Mnemonic: LoNG listing

Option: -l

Default: FALSE

char _FSL_

Mnemonic: Force Sector Length

Option: -s

Default: FALSE

char _SEF_

Mnemonic: Save Erased Files

Option: -u

Default: FALSE

char _FDC_

Mnemonic: Fix Data Character

Option: -f

Default: FALSE

char _SMS_

Mnemonic: SMart Seek

Option: -S

Default: FALSE

char _ASC_

Mnemonic: ASCii dump

Option: -c

Default: FALSE

Of course you do not get such a complete listing just for the fun of it. It can be very useful if you want to know which settings are actually in effect. The '-v' option has been added to do just that. If you issue:

```
bdde -l -Fbdde.dmp -i3 -M0 -v
```

you will get something like this:

```
(DEBUG) GetOptions: Current settings at '-v' option;
uchar O_odTrk = 0
char O_DbgLvl = 2
char O_DataType = 3
char O_ConverType = 1
char O_MediaType = 0
char O_InterLeave = 2
char O_DataChar = A
char O_BetaDump [64] = bdde.dmp
char O_Spectrum [64] = z80
char O_Prompt [16] = $
char _LNG_ = TRUE
char _FSL_ = FALSE
char _SEF_ = FALSE
char _FDC_ = FALSE
char _SMS_ = FALSE
char _ASC_ = FALSE
Available memory on heap = 5734 bytes
```

The report on free memory and the Spectrum-call are only available on MS-DOS. Please note that the position of the '-v' option is important! It will ignore any options issued after the '-v' option. Later on we will show you how to produce an even more complete report.

3.4 Extended debuglevels

In the BDDE User Guide you can read about the debuglevels. However, only the levels 2 to 5 are documented. Of course the debuglevels 0 and 1 exist too. Please don't use them if you don't need them because they generate a lot of information (more than 100 kB) and slow BDDE down considerably. If you suspect BDDE to be in error they can be very handy.

Debuglevel 1 just shows the functions and their arguments. Of course the information generated by the higher debuglevels is shown too. Therefor this debuglevel is called FLOW. It just shows the flow of the program. Every argument is shown this way:

(type)name=value

The following rules apply to the representation of the values:

1. Numeric variables like (int), (unsigned int), (long), etc. are always shown in their numeric form.
2. Variables of type (char) are displayed as characters if the chances are they contain a printable value. Otherwise they are shown as numbers.
3. Character arrays are displayed as strings if the chances are they contain an ASCIIZ string. Otherwise they are shown as pointers.
4. All other datatypes are shown as pointers.
5. If an argument has not been initialized yet it is represented by the string '[undef]'. If an argument contains a NULL-pointer it is represented by the string '[null]'.
6. If a function does not take any arguments the string 'NONE' will be displayed.

Debuglevel 0 even shows even more detailed information like allocation of dynamic memory, sector translations, interleave switches and writing chunks of data to disk. All of the most critical operations are covered here. If you want to use the extended debuglevels you have to issue the '-Dx' option with the appropriate debuglevel, e.g.

```
bdde ls -D0 -l -M0 -i1 -Fbdde.dmp
```

The debuglevel goes into effect when it is encountered. If you issue it on the commandline no messages will be displayed about the initialization file or the $BDDEINIT variable. If you want to have the most complete listing you'll have to define '-D0' within the BDDE.INI or .bdderc file. Otherwise you'll have to recompile BDDE with another default for O_DbgLvl.

This is a complete listing of all debuglevel 0 messages. There is no such list for the level 1 messages. In our opinion it wouldn't be very useful.

```
(DEBUG) GetILeave: Interleave adjusted; now set to x:1
```

BDDE found another sector-ID than expected and adjusted the interleave successfully. This message may only occur once and only while reading track 0. Otherwise it may be an indication for data-corruption. This message can only be displayed if you've used the '-M1' option.

```
(DEBUG) CheckHeader: Expected Tkx/Sdy/Scz; read Tkx/Sdy/Scz in buffer at offset xxxh
```

Shows which Track/Side/Sector was read, which one was expected and how the interleave-translation algorithm performed. Not extremely interesting except when sector-ID errors occur. This message can only be displayed if you've used the '-M1' option.

```
(DEBUG) FastTrk: xxx bytes from track xx (offset xxxxh) written to 'DOSfile'
(DEBUG) CopyFile: xxx bytes from track xx (offset xxxxh) written to 'DOSfile'
```

BDDE informs you how many bytes from which position in the track-buffer have been assigned to this DOS file. Very nice when you're interested how the allocation algorithm works.

```
(DEBUG) MakeTapeHd: Checksum for header block of file 'DOSfile' is xxh
(DEBUG) CopyFile: Checksum for data block of file 'DOSfile' is xxh
```

BDDE informs you when you use the Z80 V2.x .TAP conversion what the contents of the checksum byte are. Nice when you're debugging or just curious. Will always display invalid information when you use an unregistered copy of BDDE.

```
(DEBUG) GetListFile: Memory allocated on heap at 0xXXXX to hold 'listfile'
(DEBUG) GetIniOptions: Memory allocated on heap at 0xXXXX to hold 'initfile'
(DEBUG) GetAllOptions: Memory allocated on heap at 0xXXXX to hold init-file directory
(DEBUG) GetAllOptions: Memory allocated on heap at 0xXXXX to hold $BDDEINIT
(DEBUG) ExecMount: Memory allocated on heap at 0xXXXX to hold 'mount' options
(DEBUG) ExecCommand: Memory allocated on heap at 0xXXXX to hold internal call
```

BDDE informs you that memory has been allocated on the heap by a malloc() call and shows its pointer. When freeing this memory the pointer must be the same. Memory is allocated while interpreting list- and init-files. Apart from that memory is allocated for a very brief period as temporary storage for the BDDEINIT-variable and the directory of BDDE.COM.

```
(DEBUG) GetIniOptions: Memory on heap at 0xXXXX holding 'initfile' freed
(DEBUG) GetAllOptions: Memory on heap at 0xXXXX holding init-file directory freed
(DEBUG) GetAllOptions: Memory on heap at 0xXXXX holding $BDDEINIT freed
(DEBUG) UnMountDisk: Memory on heap at 0xXXXX holding listfile freed
(DEBUG) CleanUp: Memory on heap at 0xXXXX holding internal call freed
(DEBUG) ExecMount: Memory on heap at 0xXXXX holding 'mount' options freed
```

BDDE informs you that previously allocated memory has been successfully freed. Please note the pointer-values.

```
(DEBUG) GetIniOptions: Init-file 'BDDE.INI' not found
```

Normally BDDE does not report on missing BDDE.INI or .bdderc files. However when the debug-mode is in effect is does report on missing initialization-files.

3.5 Reading a debug listing

Like we said the '-D0' option will produce a lot of information. But how can we use and interpret this information? This section will give you a complete example by using the output of a simple

```
bdde ls -l
```

command. The BDDE.INI file contains the settings:

```
-D0 -C4 -M1 -S -Espec
```

and is read in first. Now the errorlevel has been set to debuglevel 0 BDDE starts producing debug-messages. The first indicates that the dynamic memory containing the BDDE.INI has been freed.

```
(DEBUG) GetIniOptions: Memory on heap at 0xE9FC holding D:\TEMP\bdde.ini freed
```

These are the current settings of BDDE after the default initialization file BDDE.INI has been read.

```
(DEBUG) GetIniOptions: Current settings after reading init-file;
uchar O_odTrk = 0
char O_DbgLvl = 0
char O_DataType = 3
char O_ConverType = 4
char O_MediaType = 1
char O_InterLeave = 0
char O_DataChar = A
char O_BetaDump [64] =
char O_Spectrum [64] = spec
char O_Prompt [16] = $
char _LNG_ = FALSE
char _FSL_ = FALSE
char _SEF_ = FALSE
char _FDC_ = FALSE
char _SMS_ = TRUE
char _ASC_ = FALSE
Available memory on heap = 5188 bytes
```

After reading BDDE.INI, the system-variable $BDDEINIT is interpreted. This variable contains these settings:

```
-Fbdde.dmp -M0 -i1
```

This of course affects the settings. In fact, these settings accumulate and override previous settings. Note the allocation and deallocation of dynamic memory!

```
(DEBUG) GetAllOptions: Memory allocated on heap at 0xE814 to hold $BDDEINIT
(FLOW ) GetOptions: Arguments; (char*)args[0]=-Fbdde.dmp
(char*)args[1]=-i1 (char*)args[2]=-M0
(DEBUG) GetAllOptions: Memory on heap at 0xE814 holding $BDDEINIT freed
(DEBUG) GetAllOptions: Current settings after $BDDEINIT;
uchar O_odTrk = 0
char O_DbgLvl = 0
char O_DataType = 3
char O_ConverType = 4
char O_MediaType = 0
char O_InterLeave = 0
char O_DataChar = A
char O_BetaDump [64] = bdde.dmp
char O_Spectrum [64] = spec
char O_Prompt [16] = $
char _LNG_ = FALSE
char _FSL_ = FALSE
char _SEF_ = FALSE
char _FDC_ = FALSE
char _SMS_ = TRUE
char _ASC_ = FALSE
Available memory on heap = 5752 bytes
```

Now the command-line options are interpreted. Because we issued the command 'bdde ls -l' there is only the '-l' option to evaluate and consequently only the _LNG_ flag is affected. Note the call to the GetFileList() function.

```
(FLOW ) GetOptions: Arguments; (char*)args[0]=-l
(FLOW ) GetFileList: Arguments; (char)OsCmd=0 (int)Yargn=1
(char*)Yargs[0]=-l
(DEBUG) GetAllOptions: Current settings after commandline-options;
uchar O_odTrk = 0
char O_DbgLvl = 0
char O_DataType = 3
char O_ConverType = 4
char O_MediaType = 0
char O_InterLeave = 0
char O_DataChar = A
char O_BetaDump [64] = bdde.dmp
char O_Spectrum [64] = spec
char O_Prompt [16] = $
char _LNG_ = TRUE
char _FSL_ = FALSE
```

```
char _SEF_ = FALSE
char _FDC_ = FALSE
char _SMS_ = TRUE
char _ASC_ = FALSE
Available memory on heap = 5752 bytes
```

Now we have collected all options from all sources and can start processing. Since ‘ls‘ is a core-command and accesses the disk we ’mount’ the disk and read the first track. That means calculating where we are and start reading sixteen sectors. Since there are no sector headers, we won’t have to check them.

```
(FLOW ) Execute: Arguments; (char)OSCmd=0
(FLOW ) MountDisk: Arguments; NONE
(FLOW ) GetTrk: Arguments; (unsigned char)LogTk=0
(FLOW ) LogTS_to_AbsSc: Arguments; (unsigned char)LogTk=0
(unsigned char)LogSc=0
(FLOW ) LogTk_to_FysTk: Arguments; (unsigned char)LogTk=0
(FLOW ) LogTk_to_FysSd: Arguments; (unsigned char)LogTk=0
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xC808 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xC908 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xCA08 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xCB08 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xCC08 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xCD08 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xCE08 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xCF08 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xD008 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xD108 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xD208 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xD308 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xD408 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xD508 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xD608 (int)No=256
(FLOW ) FillBuf: Arguments; (unsigned char*)buf=0xD708 (int)No=256
```

Our track-buffer now contains the contents of the first track. Now we can start to execute the ‘ls‘ command, which means constructing a directory.

```
(FLOW ) MakeDir: Arguments; (char)Cmd=0
```

This we have to do for all 14 entries: encode the BetaDisk directory entry to our internal directory, check the integrity of the entry and see if it’s on the filelist. Note that in order to display a "long list" some additional calculations have to be made!

```
(FLOW ) MakeEntry: Arguments; (unsigned char*)BDirp=0xC808 (char)Command=0
(FLOW ) CheckLength: Arguments; (unsigned char*)BDndx=0xC808
(unsigned char)DirEnt=0 (unsigned)AbSec=52
(FLOW ) CheckList: Arguments; (char*)BFname=index
(FLOW ) DirInfo: Arguments; (unsigned char)Entry=0 (unsigned)Address=9232
(unsigned)Length=9232 (unsigned char)NoSect=37
(FLOW ) LogTS_to_AbsSc: Arguments; (unsigned char)LogTk=3
(unsigned char)LogSc=4
```

```
(FLOW ) LogTS_to_AbsSc: Arguments; (unsigned char)LogTk=1
(unsigned char)LogSc=0
(FLOW ) LogTk_to_FysSd: Arguments; (unsigned char)LogTk=1
(FLOW ) LogTk_to_FysTk: Arguments; (unsigned char)LogTk=1
.
.
.
(FLOW ) MakeEntry: Arguments; (unsigned char*)BDirp=0xC8F8 (char)Command=0
(FLOW ) CheckLength: Arguments; (unsigned char*)BDndx=0xC8F8
(unsigned char)DirEnt=13 (unsigned)AbSec=412
(FLOW ) CheckList: Arguments; (char*)BFname=<0>
(FLOW ) DirInfo: Arguments; (unsigned char)Entry=13
(unsigned)Address=33150 (unsigned)Length=317 (unsigned char)NoSect=2
(FLOW ) LogTS_to_AbsSc: Arguments; (unsigned char)LogTk=25
(unsigned char)LogSc=12
(FLOW ) LogTS_to_AbsSc: Arguments; (unsigned char)LogTk=25
(unsigned char)LogSc=11
(FLOW ) LogTk_to_FysSd: Arguments; (unsigned char)LogTk=25
(FLOW ) LogTk_to_FysTk: Arguments; (unsigned char)LogTk=25
```

We're all done now, so the only thing we have to do is 'unmount' the disk and exit.

```
(FLOW ) UnMountDisk: Arguments; NONE
```

3.6 (Re)compiling BDDE

BDDE is written in C and should be portable to any platform that supports such a compiler. This includes most Unix and MS-DOS compilers.

If you want to compile BDDE under MS-DOS then follow these steps. First copy these files to your INCLUDE directory (that's where all your *.h files are):

```
EASYC.H
VERSION.C
```

Then copy all the other *.c files from the \ms-dos and \utils directory to any other directory. Now make the latter directory your current directory and compile BDDE. Check your compiler documentation for details. You do not have to make a MAKE file or anything like that.

Finally, do *not* define UNIX, HANSOFT, or DEMO. You won't get a fully functional copy of BDDE or its utilities. If your compiler supports ANSI-C we recommend that you define ANSI_C.

If you want to make a Unix-version of BDDE then follow these steps. First copy BDDE.CPI to your Unix-environment. Now unpack the files by issuing this command:

```
cpio -icvB < BDDE.CPI
```

Now copy these files to your /usr/include directory (that's where all your *.h files are):

```
easyc.h
version.c
```

Now return to the directory that contains all other BDDE C-sources and make this directory your current directory. Now type:

```
cc -DUNIX -o bdde bdde.c; cc -DUNIX -o tw2c tw2c.c
```

Both programs will now compile to ‘bdde‘ and ‘tw2c‘. Finally, do *not* define HANSOFT or DEMO. You won’t get a fully functional copy of BDDE or its utilities. However, if your system supports ANSI-C we recommend that you issue this command:

```
cc -DUNIX -DANSI_C -o bdde bdde.c; cc -DUNIX -DANSI_C -o tw2c tw2c.c
```

BDDE can produce a simple warning concerning GetAllOptions(). You can ignore it. If BDDE still doesn’t compile add this line to ’bddelib.h’:

```
#define void
```

Now everything should be alright.

Chapter 4

BDDE Utilities

4.1 Introduction

This package also includes two small utilities. I developed them because simply because I needed them. May be you need them too and I'm glad to share them with you.

4.2 BetaSpec

BetaSpec is a very small utility: about 25 bytes of Z80 machine-code. Still it enabled me to convert some programs more easily. Even better: if I had *not* used BetaSpec some programs might not have been transferred at all. Why? What is the magic of that small program. Well, no magic at all. It is just a matter of memory layout..

Some BASIC programs that I converted didn't work. It was quite easy to find out why: they had embedded machine-code. Some programmers integrate their machine-code in a BASIC program (mostly in a REM statement). The best way to call relocatable machine-code in a BASIC program is to use this statement:

```
RANDOMIZE USR ((PEEK 23635+256*PEEK 23636)+5)
```

However, some don't. The only thing you have to do is to find the statement where the code is called and change the address that is called from the USR statement. That's fairly easy.

Other code *isn't* relocatable. That's tricky and you really need a disassembler to fix this. That is, if the program isn't too large and you're really attached to it.

But why? Well, BASIC memory shifts if you add devices like an Interface one or a BetaDisk. They need RAM storage for their variables and buffers. The problem is that this memory is allocated somewhere between the system-variables and the BASIC. So the starting address from the BASIC shifts..

That is where BetaSpec comes in. BetaSpec scans the memory-layout and tries to recreate the BetaDisk-environment. This is done by calling a ROM-routine (MAKE_ROOM located at #1655). This is the routine the Spectrum uses itself to make room. Several other programs use this trick too, e.g. the famous HiSoft BASIC compiler.

Since the addresses are the same as before the program works fine, even if it uses non-relocatable code. But there are a few catches:

- The program does not work if BASIC is already located at 23867 or above.

- If any program tries to use channels (e.g. 64 characters per line routines) and you have another channel (e.g. 't') still open.
- If you have some queer kind of BetaDisk, PROG (system-variable) might not point to 23867. You can find out by typing "PRINT PEEK 23635+256*PEEK 23636". You can fix this by changing 59 into "PEEK 23635" and 93 into "PEEK 23636" in the BASIC listing.

4.2.1 How to use BetaSpec

The easiest way to use BetaSpec is to type in the BASIC-listing. If you use G.A. Lunters excellent emulator you can also use the BETASPEC.Z80 file. The BetaSpec is ready for use. If you want to run the program yourself, read in the BETASPEC.BAS file by assigning the RS232 port to BETASPEC.BAS and typing:

```
LOAD *"b"
```

Users of other emulators have to type in the listing of BETASPEC.BSC. Sorry, but it won't take that long. The machine-code is loaded in the printer-buffer (23296), but you can easily change that to any address you like. It is fully relocatable and you'll only need to run it once.

The REM in line 50 is there for security reasons. You can't run the code by accident. Remove the REM and it is fully armed. Then issue:

```
RUN
```

This could be shown:

```
CHANS: 23729 PROG : 23813
CHANS: 23846 PROG : 23867
```

The first two lines show the addresses of the channel and the BASIC areas before BetaSpec had been run, the last the same address, but after the execution of the program. If CHANS isn't exactly the same you could have trouble running programs that create extra channels.

Don't worry if you accidently run BetaSpec again: it has its safeguards built-in. You can now load your machine-code infected BASIC programs and run then *without* modification. Remember that you have to RUN BetaSpec again after you have issued a NEW.

4.2.2 Where to use BetaSpec

BetaSpec should run on any ZX Spectrum, emulated or not. Period. You might find it very handy. I've come to like it myself. What a difference 25 bytes make.. ;-)

4.3 TW2C

TW2C stands for "Tasword II converter". It's nice although it has its restrictions. It's a lot older than BDDE. That's because I've managed to pull a few text-files from BetaDisk in 1990 using some obscure program. It wasn't half as powerful as Anadisk. However, Tasword II files have a very strange format. Every line is 64 bytes and no EOL markers like LF or CR. Basically that's what TW2C does: read 64 bytes at a time and add EOL markers till it reaches the end of file. But I added a few other features.

4.3.1 Other features

Like every other word-processor Tasword II adds it own control-codes. TW2C replaces them with a space. Tasword II uses a *lot* of spaces. Like I said, every line is 64 characters long. So if you tab spaces are added. If you use a margin Tasword II adds spaces. It can get very tiresome to remove them when you read raw converted Tasword II files into a real word-processor. So TW2C removes all spaces in a line until two words are separated by only one space. If you want to convert tables, you might not want the spaces to be removed. Add the -r option and TW2C will leave them alone.

4.3.2 How to use TW2C

Let's say you have two files, one called 'letter.cod' containing a letter (surprisingly) and other called 'table.cod'. You have converted these files using BDDEs option '-C3' convert 'letter.cod' by typing:

```
tw2c -C3 letter.cod letter.txt
```

And 'table.cod' by typing:

```
tw2c -C3 -r table.cod table.txt
```

Sorry, no wildcards. That would spoil the portability. You can use MS-DOS or Unix 'FOR' command to fix that, e.g.:

```
for %%a in (*.cod) tw2c %%a subdir\%%a
```

The files created are formatted conforming the operating system TW2C has been compiled to (I love portability!). The '-Cx' option is exactly the same as the one used by BDDE. So if you converted a file using BDDEs '-C4' option, add '-C4' to TW2C! Could it be easier? Furthermore, TW2C will report how many lines it converted and how many bytes were skipped. If you use conversion 0, 1, 2 or 3 it should report that no bytes were skipped (0 bytes skipped). If you use conversion 4 it should report that 1 byte was skipped.

4.3.3 Contents

TW2C	The TW2C executable
TW2C.C	The TW2C C-source
*.C	Library functions
TW2CLIB.H	Interface file
BETASPEC.LST	BetaSpec assembler listing
BETASPEC.BSC	BetaSpec BASIC source (ASCII-format)
BETASPEC.BAS	BetaSpec BASIC source (RS232-format)
BETASPEC.Z80	Memory image of BetaSpec for G.A. Lunters Emulator

Chapter 5

History

5.1 BDDE V1.1b

- BDDE incorrectly complains about an invalid number of files when erased files are present on disk. This error had no major implications.
- The registered version now recognizes all interleaves up to 7:1. Newer versions of Anadisk sort the sectors on ID, thus effectively creating an interleave of 1:1.
- Automatic interleave detection can be overridden by the -i option. Valid arguments are -i1, -i3, -i5 and -i7. The -i option fixes the interleave and disables automatic correction.
- Released 1992-06-18

5.2 BDDE V1.2a

- A serious error occurred when the translation from BetaDisk filename to DOS- or Unix-filename resulted in an empty string. This prevented the file from being extracted. Now 'noname' is assigned to these files.
- When the automatic DATA type detection failed a BASIC file was written instead of a CODE file.
- The -H0 option replaces the -r option. -H1 (default) writes a RS232 header; -H2 writes a tapeheader. Experimental version in preperation of Atari-ST emulator support.
- Released 1992-08- 27

5.3 BDDE V1.2c

- If a target-directory was specified, the 'noname' assignment failed. Again, the file would not be written. Filename translation is now more solid. Case-sensitive filenames in Unix-version have been dropped; now all filenames are converted to lower case in both DOS- and Unix-versions.
- A trailing separator can be added to the target-directory.
- Atari-ST emulator support is complete. Spectrum-filenames in tapeheader and DOS/Unix filenames are now synchonized. Changed -Hx option to -Cx option.
- All documentation now available in English.
- Released 1992-09- 24

5.4 BDDE V1.3c

- Complete SpecEM (another Spectrum Emulator by Kevin Phair) support by adding -C3 option.
- Now prints autoRUN linenumber.
- In earlier versions BDDE incorrectly complained about BASIC-files that allocated too many sectors. This has been corrected.
- BDDE now reports when a file has to be renamed when switching to 'tempnames' filename generation.
- A distinction is made between character- and number DATA-files before the filename is generated. This was necessairy for SpecEm-type filename generation but has also been added to the Lunter conversion.
- Adding support for other emulators has been significantly simplified.
- Released 1992-10-27

5.5 BDDE V1.4a

- Added .TAP file support for Z80 V2.x emulator by Gerton Lunter by adding -C4 option.
- Added BetaDisk+ snapshot detection.
- Added support for interleave 2:1, 4:1, 6:1 and 8:1
- File allocation error did not allow stringing .TAP files together. This has been permanently fixed.
- Released 1993-09-17

5.6 BDDE V1.5c

- Added several functions to monitor BDDEs internals.
- Added K&R C support.
- Syntax drastically changed to accomodate several new features.
- Added conditional extraction.
- Added '-l' option to simplify constructing listfiles.
- Added listfiles.
- Added BDDEINIT- and BDDE-variables.
- Added automatic and manual initialization files.
- BDDE can now also read raw dumps (without sector headers).
- BDDE now checks whether the file is a valid dump.
- BDDE now also available under Coherent V4.2.
- Released 1994-08-22

5.7 BDDE V1.7c

- Added a tiny desk calculator with the ‘dc‘ command
- Changed the format of the error-messages
- Released a MS-Windows version
- Added an interactive shell (some commands are only available to registered users)
- Added wildcard support
- Added switches to disable certain options, e.g. -S, -s, -l
- Released 1994-12-07

5.8 BDDE V1.7e

- Removed the EasyC syntax, now ordinary C
- Added support for several C compilers
- Released a Windows 32-bit version
- Added an installer to the Windows version
- Released a Linux version
- Ported the entire documentation to LyX
- Placed the entire package under GPL
- Released 2002-06-10