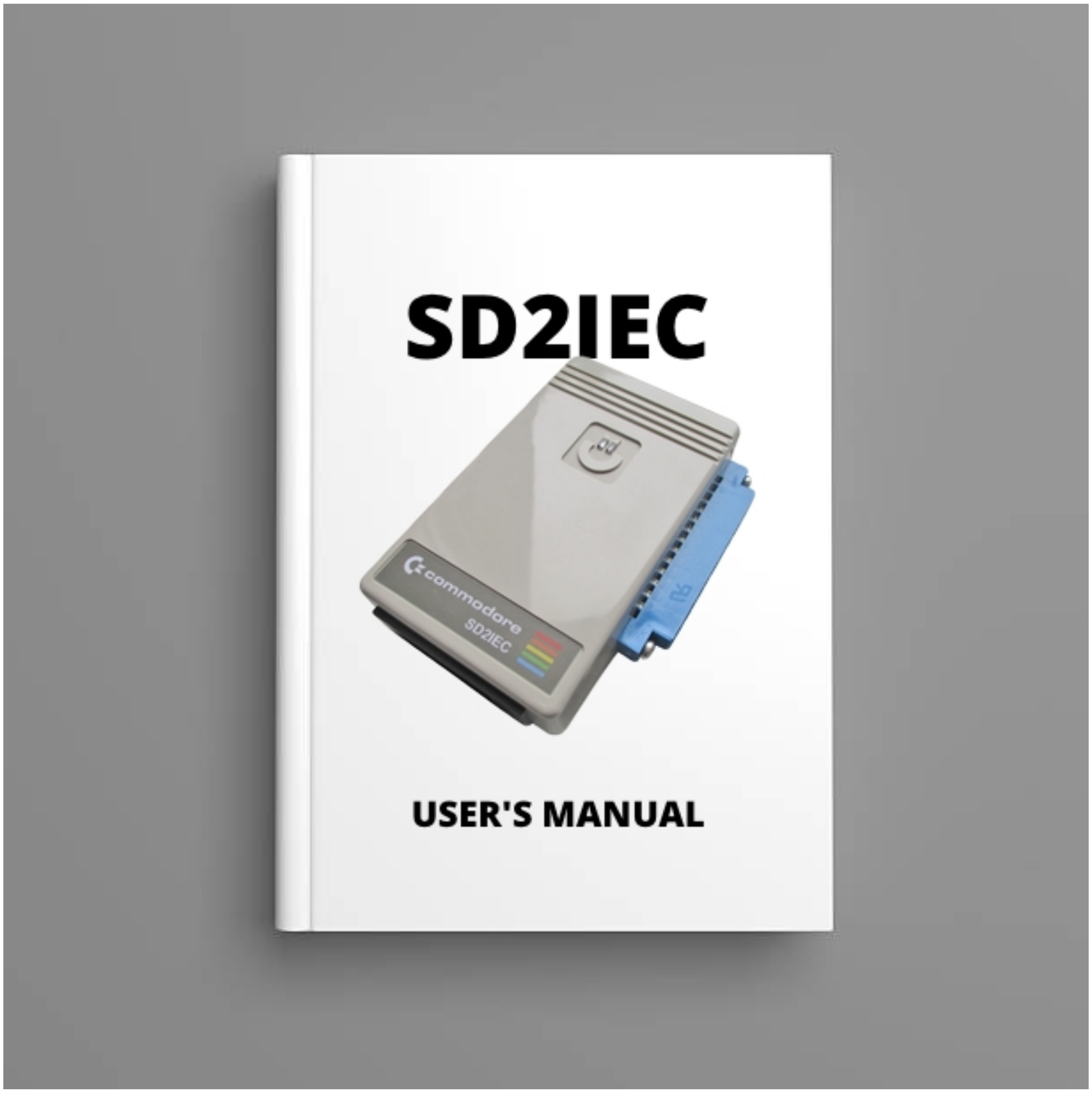
SD2IEC User's Manual



Document Version: 1.2

There are many SD–Card–based storage solutions for Commodore 8–bit computers today, virtually all of which are based on the sd2iec firmware. The hardware comes in a variety of packages, with different buttons, cases and means of obtaining power. Some go under different branded names, and some include special features such as an LCD display, a daughter board or an internal mounting bracket. But they almost all run some variant of the same software. And it is the software inside the devices that make them feel and behave alike when using them on your C64.

sd2iec is firmware, used in hardware designs like MMC2IEC, SD2IEC, and uIEC. It allows the Commodore serial bus to access removable storage devices (MMC, SD, CF). Think of SD2IEC as a CMD HD but with a modern storage medium instead of a SCSI harddrive.

This User's Manual covers the sd2iec firmware, its commands, features, configuration, software support and limitations. It does not cover hardware device specific features. There are some generic hardware features that the firmware interacts with. These are covered, but they may or may not be present on your particular hardware device. Examples include: a reset button, swap button, next and previous buttons and device address jumpers or switches.

SD2IEC Hardware Devices

Check the Commodore 8-Bit Buyer's Guide: Storage for more information about the assortment of SD2IEC hardware devices currently on the market.

Some common sd2iec–based, SD2IEC hardware devices.

NOTE: To distinguish the hardware from the firmware, hardware devices which use the firmware are referred to in this User's Manual in uppercase: SD2IEC. References to the firmware are in lowercase: sd2iec.

The sd2iec command set and file and device access features are based upon Commodore DOS that was developed for Commodore's original floppy disk drives, and was used and popularized by the widely sold 1541. sd2iec includes other features that were added by later Commodore disk drives and by the CMD line of storage devices, (CMD FD, CMD HD and CMD RamLink). Building upon the command syntax of Commodore DOS helps to maintain compatibility with existing software, although, compatibility is not complete.

sd2iec supports the majority of features available to earlier disk drives, and adds many new features to take advantage of the more advanced hardware. Although some DOS commands are not available in sd2iec, the biggest reason for compatibility issues is that the original DOS ran on the drive's own stripped down 6502–based computer. Commodore DOS exposed implementational details that allowed programmers—encouraged them even—to execute code on the drive. This was taken advantage of by many speed loaders. Such software may encounter compatibility issues with sd2iec, which is not based on a 6502 microprocessor, and cannot run drive specific code.

While this User's Manual goes into some detail, it should be considered a supplementary document to more complete user manuals such as for the 1541 or CMD storage devices. These manuals and many others can be found online, at commodore.bombjack.org, and should be referred to for further detail. To understand the directory path format, how to interpret command syntax, how to use the error channel and more, *Section 9: Command Reference* of the CMD Hard Drive User's Manual should be considered essential reading.

Important Reference Material

- 1541–II Disk Drive User's Guide
- 1581 Disk Drive User's Guide
- CMD FD–Series Disk Drives User's Manual
- CMD Hard Drive User's Manual

This User's Manual is based upon the README found in the sd2iec git repository. Supplementary material is sourced from the above–linked user manuals. Some content is repeated verbatim, other content has been modified and expanded upon.

EDITOR'S NOTE: There are a handful of *TODO* notes throughout the manual. These represent gaps in the author's current knowledge, and further testing is required. These notes will be eliminated and replaced with meaningful commentary as the testing is completed.

Table of Contents

Files Table of Contents

Files are the most common way to organize data on a storage medium. A typical Commodore file consists of a stream of data and an associated directory entry. The directory entry records its name, file size measured in 256–byte blocks, creation date and Commodore file type, as well as some status flags and a pointer to where the start of the data can be found.

There have always been a few sticking points and incompatibilities when transferring files between a Commodore computer and a PC or Mac.

SD2IEC allows you to move an SD Card back and forth between your Commodore and other computer platform. This ability to share a single storage medium between different computing platforms makes transferring files from the outside world to your Commodore computer faster, easier and more convenient than ever before. However, it also brings into relief the places where incompatibilities exist.

With the development of emulators and other cross–platform tools, a number of solutions have arisen to bridge the gaps between our beloved platform and the rest of the world. sd2iec implements support for many of these solutions.

Long Filenames

Commodore DOS filenames can be 16 characters long, including any filename extension. A filename extension is optional.

sd2iec SD Cards are formatted with the FAT file system. FAT originally only supported filenames with 8 characters plus a 3 character extension. Long filenames (i.e names not within the 8.3 limits) are supported on FAT, but for compatibility reasons the 8.3 name is used if the long name exceeds 16 characters.

If you use anything other than ASCII characters or their PETSCII equivalents when editing the filename on a PC or Mac, you may get strange characters on the Commodore system. The reason is because the long filename uses unicode characters on disk, but sd2iec parses only the low byte of each character in the name.

x00 files

The x00 family of file formats was originally designed for the PC64 emulator. Because Commodore DOS supports features of filenames that will not necessarily be supported by the file systems used by host operating systems, this wrapper format was devised.

The original file is prepended with a fixed–length, 26 byte header. The header contains the original 16 character filename. The file is then named following the conventions required by the host file system. One of 4 file extensions are used for each of the 4 Commodore DOS file types.

P00	PRG
S00	SEQ
U00	USR
R00	REL

It may occur that more than one file with 16 character Commodore DOS filenames result in multiple files in the host file system that would bear the same name. This would happen for instance, if the first 8 characters of the Commodore filenames were the same, and only these first 8 characters were used for a FAT 8.3 filename. In this case, the files can be distinguished from one another in the host file system by incrementing the extension's number from x00 to x01, x02 and so on up to x99. In practice, it is uncommon to see files with an extension number greater than 00.

P00/S00/U00/R00 files are transparently supported by sd2iec. They display in the Commodore directory listing with their internal filename instead of the FAT filename. Renaming them only changes the internal name. The XE command (See: Settings) defines if x00 wrappers are used when writing files.

By default sd2iec uses them for SEQ/USR/REL files but not for PRG. Parsing of x00 files is always enabled even when writing them is not.

x00 files are recognized by checking both the extension of the file (P/S/U/R with a two-digit suffix) and the header signature.

Relative files

Relative files, Commodore DOS type REL, use a special non–sequential block layout. This file type is uncommon, but useful for implementing database–style records in a single file.

Partial relative file support is implemented. It should work fine for existing files, but creating new files and/or adding records to existing files may fail. Relative files in disk images are not supported yet, only as files on a FAT medium. When x00 support is disabled the first byte of a relative file is assumed to be the record length.

How relative files are structured is a complex topic, and for complete details you should refer to the 1541–II Users Guide, Chapter 6.

M2I files

DEPRECATION NOTICE
Support for M2I files will be removed in the next release, see the deprecation notices at the end of this document for advice.

M2I files are fully supported. sd2iec supports SEQ and USR files in this format in addition to PRG and DEL which were already implemented in MMC2IEC. For compatibility reasons the file type is not checked when opening files. Inside an M2I file the files are always shown as 0 (DEL) or 1 blocks because calling stat for every file was slowing down the directory listing too much.

For compatibility with existing M2I files the data files do not use P00 headers even when the file type is SEQ or USR.

Command Reference

File commands are the most commonly used commands. They include loading and saving files, verifying, renaming, scratching copying and locking files and are entered in the same manner as for other Commodore or compatible disk drives. Although the BASIC 2.0 versions of these commands are supported in BASIC 7.0, BASIC 7.0 also contains other commands that perform the same functions. Note that the BASIC 2.0 versions of these commands allow more versatility when dealing with partitions.

Loading Files

The following syntax can be used to load programs in BASIC 2.0 and BASIC 7.0:

```
LOAD"[[n][path]:]pattern",dv[,sa]
```

Where:

Variable	Description
n	the partition which holds the file (1 to 254)
path	the subdirectory path to the file
pattern	the name of the file or a pattern matching the file, up to 16 characters
dv	the current device number of the sd2iec
sa	the secondary address if needed

See Pattern Matching for a discussion of filename pattern matching.

To load a machine language program, a secondary address of 1 must be added to the end of this command, separated from the device number by a comma.

Examples:

```
LOAD"2:BASEBALL",12
LOAD"3/TERMS/:TERMBOOT",12,1
```

JiffyDOS Examples:

```
/"2:BASEBALL",12
/2:BASEBALL
%3/TERMS/:TERMBOOT
```

It is generally a good idea to use BASIC 2.0 syntax if you are specifying partitions since BASIC 7.0 will only allow access to the current (0) partition or partition 1, and will not allow the use of subdirectory paths.

The BASIC 7.0 BLOAD command can be used to load machine language or data files into memory. The DLOAD command is used primarily to load BASIC programs. The syntax for these commands is as follows:

```
BLOAD"filename"[,Dn][{ON|,}Udv][,Bb][,Pa]
DLOAD"filename"[,Dn][{ON|,}Udv]
```

Where:

Variable	Description
filename	the name of the machine language program to be loaded
n	the partition which holds the file (0 or 1)
dv	the current device number of the sd2iec

b	the memory bank where the file is to be loaded
a	the starting address of the file to be loaded

Examples:

```
BLOAD"SPRITE",D0,U9,B0,P3584
DLOAD"TEST",D0 ON U9
DLOAD"TEST2"
```

Saving Files

The following syntax can be used to save programs in BASIC 2.0 and BASIC 7.0:

```
SAVE"[[@][n][path]:]filename",dv
```

Where:

Variable	Description
n	any legal partition number from 1 to 254
path	the subdirectory path where the file is to be saved
filename	any legal filename of up to 16 characters
dv	the current device number of the sd2iec

The "@" symbol shown in the command syntax may be used to indicate that a file with the same name which already exists should be replaced with the new file. This is called the "Save with Replace" option and if it is used must be followed by a partition number and a colon (:). Note that in Commodore and CMD DOS, the old file is not deleted until after the new file is successfully saved. sd2iec does not have this safety feature. sd2iec deletes the original file first, and then saves the new file. Losing power during a

"Save with Replace" operation could result in a loss of data.

To save a machine language program from the C64 or c128 in 64 mode, you must use a machine language monitor or change some of the BASIC pointers.

Examples:

```
SAVE"2:BASEBALL",12
SAVE"/TERMS/:TERMBOOT",12
```

JiffyDOS Examples:

```
←"2:BASEBALL",12
←/TERMS/:TERMBOOT
```

You may use the BASIC 7.0 BSAVE and DSAVE commands when it is your intention to work with the current partition (0) and directory, or in the current directory of partition 1. BSAVE is intented for files other than BASIC programs. (The B is for Binary, not BASIC). DSAVE is intended for BASIC programs (The D, in this case, is for Disk, or any IEC device, as opposed to Tape.)

```
BSAVE"[@]filename"[,Dn][{ON|,}Udv][,Bb],Pa TO Pe
DSAVE"[@]filename"[,Dn][{ON|,}Udv]
```

Where:

Variable	Description
filename	the name of the file to be saved

n	the partition number where the file is to be saved
dv	the current device number of the sd2iec
b	the memory bank where the file to be saved is
a	the starting address of the file to be saved
e	the ending address of the file to be saved

The "@" symbol may be included to indicate that the file being saved is to replace an existing file with the same name. This is called "Save with Replace."

Examples:

```
BSAVE"SPRITE",D0,U9,B0,P3584
DSAVE"TEST",D1 ON U9
DSAVE"TEST 2"
```

Verifying Files

BASIC 2.0 and 7.0 contain commands which allow you to verify if a program has been saved properly. These commands compare the saved program with the contents of memory. Keep in mind that any change in the contents of memory may cause a verify operation to fail. For this reason, it is best to verify a file immediately after saving it. Both versions of BASIC support specifying a partition within the filename portion of this command (as described earlier). The following syntax applies to this commands:

```
VERIFY"[[n][path]:]filename",dv[,sa]
```

Where:

Variable	Description

n	the partition number where the file is located
path	the subdirectory path to the file you wish to verify
filename	the name of the file to be verified
dv	the current device number of the sd2iec
sa	secondary address of 1 to verify a non–BASIC file

Examples:

```
VERIFY"NEWSTATS",12
VERIFY"1/UTILS/TERMS/:XLATOR",12
```

JiffyDOS Examples:

```
'"NEWSTATS",12
'1/UTILS/TERMS/:XLATOR
```

In BASIC 7.0, the standard VERIFY command is accepted, but you may also use the DVERIFY command. This command is limited to use with the current partition (0) or partition 1, and the current subdirectory.

```
DVERIFY"filename"[,Dn][{ON|,}Udv]
```

Where:

Variable	Description
filename	the name of the file to be verified
n	the partition number where the file is located (0 or 1)
dv	the current device number of the sd2iec

Examples:

```
DVERIFY"NEWSTATS",D1,U12
```

Renaming Files and Subdirectories

Filenames and subdirectory names may be changed using either the DOS RENAME or the BASIC 7.0 RENAME command. The BASIC 7.0 version only supports the current directory within the current partition (0) or partition 1. When using either version, the partitions specified for the two filenames must either be the same, or must indicate the same partition. The syntax for the DOS RENAME command is:

```
OPENlf,dv,15:PRINT#lf,"R[n][path]:newname=[[n]
[path]:]pattern":CLOSElf
```

Where:

Variable	Description
lf	the logical file number for the command channel
dv	the current device number of the sd2iec
n	the partition where the file to be renamed is located
path	the subdirectory path to the file to be renamed
newname	the new name to be assigned to the file
pattern	the name of the file or a pattern matching the file which is being renamed *

** If the pattern matches more than one file, only the first matched file will be renamed.*

Renaming files works the same as it does on CMD drives, although the errors flagged for invalid characters in the name may differ.

See Pattern Matching for a discussion of filename pattern matching.

Examples:

```
OPEN15,12,15:PRINT#15,"R1:BOOT1=BOOT":CLOSE15
OPEN15,12,15,"R1/UTILS/:NEWT=1/UTILS/:WW":CLOSE15
```

JiffyDOS Examples:

```
@"R1:BOOT1=BOOT",12
@R1/UTILS/:NEWT=1/UTILS/:WW
```

The BASIC 7.0 RENAME command syntax is:

```
RENAME[Dn,]"filename"TO[Dn]"newfile"[,Udv]
```

Where:

Variable	Description
n	the partition where the file to be renamed is located (0 or 1)
filename	the name of the file which is being renamed
newfile	the new name to be assigned to the file
dv	the current device number of the sd2iec

Example:

```
RENAME "myoldfile" TO "mynewfile",U12
```

Scratching (deleting) Files

The Commodore DOS and BASIC 7.0 SCRATCH commands may be used to delete files from a partition. As with many of the other BASIC 7.0 commands, SCRATCH is only effective with partition numbers 0 (current) and 1. The standard Commodore DOS SCRATCH may be used with BASIC 2.0 and in place of the BASIC 7.0 version when necessary. The following command syntax is for the DOS version of this command:

```
OPENlf,dv,15:PRINT#lf,"S[n][path]:pattern[,[n][path]:pattern...]
```

Where:

Variable	Description
lf	the logical file number for the command channel
dv	the current device number of the sd2iec
n	the partition(s) which hold the file(s) to be scratched
path	the subdirectory path(s) to the file(s)
pattern	the name of the file(s) or a pattern matching the file(s) to be scratched

Multiple files may be scratched with this command. There is no limit on the number of files, except the maximum command length. (Usually 100 to 120 characters.) The filename parameters may also contain wildcards to allow scratching multiple matching files in the same partition. Directories are ignored.

See Pattern Matching for a discussion of filename pattern matching.

Examples:

```
OPEN15,12,15:PRINT#15,"S1:JUNK,3:C?*.BAS":CLOSE15
OPEN15,12,15,"S1/UTILS/:CO*":CLOSE15
```

JiffyDOS Examples:

```
@"S1:JUNK,3:C?*.BAS",12
@S1/UTILS/:CO*
```

The BASIC 7.0 SCRATCH command syntax is:

SCRATCH"*pattern*"[,D*n*][{ON|,}U*dv*]

Where:

Variable	Description
pattern	the name of the file or a pattern matching the file(s) to be scratched
n	the partition where the file to be scratched resides (0 or 1)
dv	the current device number of the sd2iec

Multiple files may be scratched with this command by using pattern matching, although it does not allow you to specify multiple filenames as does the DOS version. Remember, this command is only valid for use with the current directory in partition 0 (current) or 1.

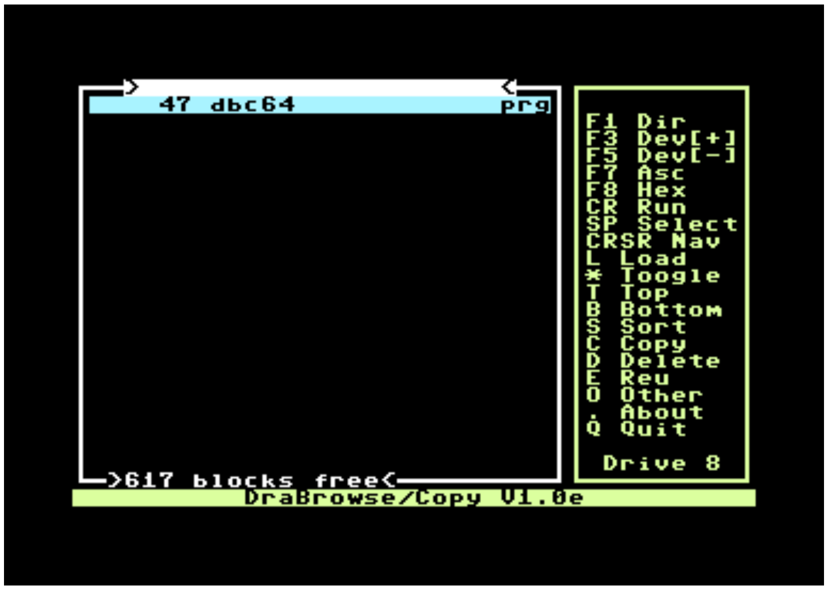
See Pattern Matching for a discussion of filename pattern matching.

Example:

```
SCRATCH"C*.BAS",D0 ON U9
```

Copying Files

Copying is an important consideration with any storage device. For this reason sd2iec has DOS commands which allow you to copy files between directories and between partitions on the same device and SD Card. This function may also be accomplished using a copying program such as DraCopy.



Files may be copied between sd2iec and other drives using a standard file copier. Only generic copiers that do not try to discover the drive type by checking ROM locations will work with sd2iec. FCOPY, MCOPY, BCOPY, WCOPY, etc. as provided by CMD for their storage devices do not work with sd2iec, as they detect sd2iec as 1541. DraCopy is a good alternative, and provides a number of other features not offered by FCOPY.

If you have JiffyDOS installed on your computer, you may also use the built–in JiffyDOS file copier with sd2iec. Many commercial copy programs will not work with sd2iec because they look at specific memory locations to identify the drive and/or they attempt to write code into the drive to speed up the copy process.

Copying and Combining Files between partitions

You may copy files from one partition to another, or between different directories on an SD Card by using the standard Commodore and CMD DOS COPY command. This command allows you to place a partition number and path in front of each of the filenames specified in the command. The syntax for this command is as follows:

```
OPENlf,dv,15:PRINT#lf,"C[n][path]:newfile=[[n]
[path]:]pattern[,[n:]pattern...]":CLOSElf
```

Where:

Variable	Description
lf	the logical file number for the command channel
dv	the current device number of the sd2iec
n	the partition which holds or is to receive the file
path	the subdirectory path(s) where the file(s) to be copied or created is (are) located
newfile	the name of the new file being created
pattern	the name of the file(s) or a pattern matching the file(s) which is (are) being copied

You can use this command to copy multiple files into a single target file in which case all source files will be appended into the target file. Parsing restarts for every source filename which means that every source name is assumed to be relative to the current directory. You can use wildcards in the source names, but only the first file matching will be copied.

Copying relative files should work, but isn't tested well. Mixing relative and non-relative files in an append operation isn't supported.

See Pattern Matching for a discussion of filename pattern matching.

Examples:

```
OPEN15,12,15:PRINT#15,"C1:FCOPY=3:FCOPY":CLOSE15
OPEN15,12,15,"C:FULLSTATS=STAT1,3:STAT3":CLOSE15
OPEN15,12,15,"C2:MCOPY=1/COPIERS/:MCOPY":CLOSE15
```

JiffyDOS Examples:

```
@"C1:FCOPY=3:FCOPY",12
@C:FULLSTATS=STAT1,3:STAT3
@"C2:MCOPY=1/COPIERS/:MCOPY"
```

You may also copy files from one partition to another on the SD Card by using the standard BASIC 7.0 COPY command. This command is limited to copying files in the current directory of the current partition (0) or partition 1. Use the DOS COPY command discussed above if you want to copy files between other partitions. The syntax for this command is as follows:

```
COPY[Dn,]"filename" TO [Dn,]"newfile"[{ON|,}Udv]
```

Where:

Variable	Description
n	the partition which holds the file to copy or to receive (0 or 1)
filename	the name of the file which is being copied
newfile	the name of the new file being created
dv	the current device number of the sd2iec

Example:

```
COPY D0,"MYFILE.TXT" TO D0,"MYFILE2.TXT",U12
```

If you use this command to copy a file to a different partition, you may use the same filename for both files. If they are to reside in the same partition, you must use different filenames or an error will result.

Two files may be combined into a single file by using the BASIC 7.0 CONCAT command, although it is important to note that adding files together in this manner is only effective with text files. The BASIC 7.0 limitation of using only the current directory of the current partition (0) or partition 1 applies to this command. The syntax is:

```
CONCAT[Dn,]"fromfilename" TO [Dn]"tofilename"[,Udv]
```

Where:

Variable	Description
n	the partition where the files exist (0 or 1)
fromfilename	the name of the file being added
tofilename	the name of the file being added to
dv	the current device number of the sd2iec

The file previously named *tofilename* will be replaced by the newly created combined file. The only way these files may have the same name is if they exist in different partitions.

TODO: Test and verify if *from* and *to* files can be in different partitions.

Example:

```
CONCAT D0,"MYFILE.TXT" TO D0,"MYFILE.TXT",U12
```

Locking and Unlocking Files

Locking files is an under–used feature that has been part of Commodore DOS for a long time. The lock flag is bit 6 of the file type byte. Commodore DOS v2.6, found on the 1541, however, does not include a command to toggle the lock flag. A third–party utility is needed to change the bit, such as Epyx Fastload or JiffyDOS. Alternatively, if you know how, you can use a sector editor to manually change the bit. The 1541–II User's Guide (page 70) lists a 12–line BASIC program that can open a direct file, manipulate the buffer pointer, read in the file type byte, toggle the lock bit, write the byte back to the buffer and write the buffer back to disk. Needless to say, it is not an easy procedure to manually lock or unlock a file.

CMD DOS added a Lock/Unlock command (*L*), which greatly improves matters for users of CMD storage devices. This command can be used on native partitions or any emulation mode partition. The behavior of the lock flag is to prevent the user from accidentally scratching a file. Unfortunately, a locked file can still be overwritten with the save–and–replace (@) feature. Additionally, in a CMD Native partition, a directory can have its lock bit set, but it doesn't protect files inside the folder. It only prevents the folder itself from being removed by the Remove Directory (*R-D*) command.

sd2iec does not support CMD DOS's Lock (*L*) command. Files and directories inside a mounted disk image can have their lock bit set manually, or with a third party utility.

TODO: Test if JiffyDOS's and/or EPYX Fastload's Lock feature works when a disk image is mounted.

Positioning (seeking) Within a File

In Commodore and CMD DOS, only relative files can be accessed non–sequentially. All other file types, PRG, SEQ and USR, in the Commodore file system uses a simple singly-linked list of blocks. The directory entry only points to the first block in the list. In order to find the location of block 5, for example, the drive must load block 1 to find the pointer to block 2. Then must load block 2 to find block 3 and so on, following the chain one block at a time. The FAT file system uses a more advanced *File Allocation Table* for identifying the blocks that make up a file.

sd2iec exploits the more advanced FAT file system by extending the positioning command to allow positioning the file cursor an arbitrary number of bytes into any file. In the C Standard I/O library, positioning is called *seeking*. The positioning command can also seek in reverse, moving the position of the file cursor backwards to an earlier part of the file. The syntax to position within any regular file is as follows:

```
OPENlf,dv,15:PRINT#lf,"P"+CHR$(ch)+CHR$(lo)
[+CHR$(mlo)[+CHR$(mhi)[+CHR$(hi)]]]:CLOSElf
```

Where:

Variable	Description
lf	the logical file number for the command channel
dv	the current device number of the sd2iec
ch	the channel of an open file in which to move the position *
lo	the (0th) low byte of a 32–bit position
mlo	the (1st) mid–low byte of a 32-bit position (optional, defaults to 0)
mhi	the (2nd) mid–high byte of a 32-bit position (optional, defaults to 0)
hi	the (3rd) high byte of a 32-bit position (optional, defaults to 0)

** According to the 1541-II User's Guide, for relative files, the channel must* *be specified as the actual channel + 96*

When used for regular files the command may take from 1 to 4 bytes of positioning, in little endian order, least significant byte first. Any high bytes omitted default to zero.

A file must already be open before using the positioning command. The command channel must be left open after issuing the positioning command, because closing the command channel closes all open files. After using the positioning command, subsequent reads from the open file will be start from the new position.

Examples:

```
OPEN15,12,15:PRINT#15,"P"+CHR$(5)+CHR$(10)+CHR$(2)
OPEN15,12,15,"P"+CHR$(8)+CHR$(0)+CHR$(0)+CHR$(3)
```

Refer to the CMD Hard Drive Users Manual, Section 9-31 for full details on relative files.

Directories Table of Contents

sd2iec provides native access to the subdirectories of the FAT file system on the storage media. The commands used to create, remove, and move around within subdirectories can be found below.

When a subdirectory is created, a DIR type file is created and added to the current directory. Subdirectory names may be up to 16 characters long. If a subdirectory is named with more than 16 characters on a PC/Mac the directory will be displayed on the Commodore system with a shortname that conforms to the 8.3 filename limits.

The storage space available to a subdirectory is the same as that available to the parent directory, the directory in which the subdirectory exists. All blocks in a partition are shared between all directories within that partition.

Subdirectories may be created in the root directory, the first or main directory in that partition, or nested within another subdirectory. There is no limit to how deeply subdirectories may be nested, but there are limits to what is convenient to work with in practice. The practical limits are based on how many characters can be passed in a single command. Usually between 100 and 120 characters, maximum. At 16 characters, plus a 1 character delimiter, plus command overhead, a nesting depth of ~7 is the reasonable practical limit, deeper if fewer than 16 characters are used for the directory names.

Loading a Directory

Just as in Commodore DOS, a directory may be loaded by loading the special *program* named "$". The directory may also be accessed as a sequential read-only file by opening a file named "$" on channel 0, the channel reserved for loading programs.

When a directory is loaded normally, it replaces any BASIC program that is currently in memory. Once loaded, a directory may be listed like any BASIC program, with the *LIST* command. Common DOS wedges, such as EPYX Fastload or JiffyDOS, allow you to list a directory directly from the drive without first loading it into memory and without disrupting the BASIC program currently in memory. The syntax for loading directories is as follows:

```
LOAD"$[[n][path][:pattern[=type]]]",dv
```

Where:

Variable	Description
n	the partition from where to load the directory
dv	the current device number of the sd2iec
path	the path to the directory to load
pattern	the pattern that matches filenames to be loaded in the directory
type	the filetype to match files to be loaded in the directory

Examples:

```
LOAD"$2:S*=P",10
OPEN5,10,0,"$1"
```

JiffyDOS Examples: (non–memory disrupting)

```
@$2:S*=P
@"$1",10
```

File Sizes and Blocks Free

As in standard Commodore DOS, file sizes in the directory are listed in 254–byte blocks. On Commodore and CMD file systems, the size of a file is only known to an accuracy of 254 byte blocks. The amount of storage used in the last block can only be determined by following the block chain all the way to the end. The FAT file system knows the size of files to single byte accuracy.

When a directory is output by sd2iec, the low byte of the BASIC next–line pointer is set to *(filesize MOD 254) + 2*. Or, the amount of storage used in the final block, plus 2. (The 2 is added so it can never be mistaken for an end marker (0) or for the default value, 1, used by at least the 1541 and 1571 disk drives.) This allows programs that read in the directory to calculate true filesizes.

The blocks free message at the end of the directory listing, however, does not actually report free blocks. On sd2iec the number reported is free clusters. A cluster, in the FAT file system, is the smallest amount of disk space that can be used to hold a file.

Cluster size is configurable when storage media is partitioned. A common cluster size is 4 kilobytes. Reporting free clusters rather than free blocks is a compromise of compatibility, accuracy and code size.

Pattern Matching

Selective directories may be loaded in the standard way, by placing a colon (:) at the end of the partition number or path, and by using a filename or pattern matching characters to determine which files to include in the listing.

The two special characters used in pattern matching are the asterisk (*) and the question mark (?). They act something like a wild card in a game of cards. The difference betweeen the two is that the asterisk makes all characters in and beyond its position wild, while the question mark only makes its own character position wild. Only one asterisk may be used in a pattern. And may be combined with one or more question mark. Here are some examples, and their results:

"A*" matches all files that begin with "A", regardless of what follows. "ARTIST", "ARTERY", and "AZURE" would all qualify, but "BARRY" wouldn't, even though it has an "A" elsewhere in its name.

"SM?TH" matches all files that start with "SM", end with "TH" and have any one character between them. This would match "SMITH" or "SMYTH", but not "SMYTHE".

"B?T*" matches all files that start with "B", and have "T" as the third character. The asterisk will match 0 or more characters. Thus, this pattern will match both and "BIT", "BYTE".

NOTE: The asterisk (*) and question mark (?) cannot be used as regular characters in a filename.

Directory Filters

The equals (=) sign and a file type designator may also be included after the filename or filename pattern to indicate a particular file type. The file type characters are:

P	for program	PRG
S	for sequential	SEQ
U	for user	USR
R	for relative	REL
B*	for branch	DIR
D	for directory	DIR
H	to show hidden files	...H

** If filtering a directory programmatically, B should be used for compatibility with CMD storage devices. D is only supported by sd2iec. On all other drives, D matches all file types.*

The FAT file system has a special *hidden* property that can be set on individual files and subdirectories. To include hidden files in the directory, use H in place of the directory filter file type character. The Commodore FS, and the extended Commodore FS used on CMD storage devices do not support hiding files. On Commodore and CMD drives the H filter character has no effect.

sd2iec marks hidden files with an H after the lock mark, which comes after the file type. If the file is not locked, a space is left where the lock mark would go. i.e. "PRG<H" or "PRG H".

Time and Date Stamped Directory Listings

Not all SD2IEC devices support a hardware Realtime Clock. For devices that do, a time and date stamp are applied to files when they are saved or replaced. The FAT file system supports time and date stamps on files, and even if your SD2IEC does not have an RTC, files that were created on a PC/Mac will have time and date stamps.

See Realtime Clock section below, for commands to read and write to the SD2IEC device's RTC hardware.

The following options allow the stamp to be viewed, and also permit the user to select files which were created within a specified timeframe. The syntax for the time and date stamped directory is:

```
LOAD"$=T[n][path][:pattern[={tp|option}[,option]]]",dv
```

Where:

Variable	Description
n	the partition number of the directory to be loaded
path	the subdirectory you wish to view

pattern	name of file or pattern to match
tp	first character of file type (P,S,R,U or B/D)
option	one of the options listed below
dv	the current device number of the sd2iec

Options:

Variable	**Description**
L	long format
N	Do not include time and date in listing
>stamp	greater than or equal to *stamp*
<stamp	less than or equal to *stamp*

```
stamp format:    MM/DD/YY HH:MM xM
```

Although the syntax for this command may look a little complex, it is really quite simple to use when broken down into separate elements.

The partition number (*n*) may be specified if desired. If this parameter is omitted, the current partition will be targeted for this command.

The *pattern* is the name of a file or a standard pattern matching string. This means you may use the asterisk (*) to match a number of characters, and question marks (?) to match individual characters. See Pattern Matching for examples.

The file type (*tp*) is optional, but if specified it must be the first option after the filename pattern. See Directory Filters for examples. If you wish to view all file types, skip this option.

The *options* allow you to match specific times and dates (>stamp, <stamp), and also to specify long format (L). You may also specify that the directory

entries match a certain time and date stamp, but that the directory list is not to include these times and dates (N). You may use as many of these options as you wish, but they must be separated by commas (,).

The *long* time format gives the full date and time:

```
112  "TESTFILE"        PRG   07/27/19 03.44 PM
```

The *short* (default) time format gives the date and time as follows:

```
112  "TESTFILE"        P 07/27 03.44 P
```

If the no–list (N) option is given, the directory entries will be loaded as they normally appear whether the long format (L) is specified or not. This means that specifying the long format and the no–list option in the same command is usually a waste of time. The reason that the no–list option was created was to allow you to use the time and date of files as pattern matching criteria within programs which cannot accept the extra time and date characters in the directory listing.

The *<stamp* option will list all files which have a creation time and date less than or equal to the time and date specified in *stamp*. The *>stamp* option will list all files which have a creation time and date greater than or equal to the time and date specified in *stamp*.

If both *<stamp* and *>stamp* options are used within the same command, the resulting list of files will include files which fall between the range of the two time and date stamps specified.

The *stamp* format must be entered exactly as shown. This means you must specify the month, day, and year with two characters each and separate them with a slash (/). The hour and minute must also be given with two characters each in 12 hour format separated with a colon (:) or a period (.). The last parameter must be AM or PM. The date and time must be separated by a single space, and so must the time and AM/PM parameters. Here are a few examples:

```
LOAD"$=T",12
LOAD"$=T2",12
LOAD"$=T2:*=P",12
LOAD"$=T2:*=P,L",12
LOAD"$=T2:*=P,L,>12/21/18 04:15 PM",12
LOAD"$=T:*=L,<12/21/18 04:15 PM",12
LOAD"$=T4:*=S,N,>12/21/18 12:01 AM,<12/31/18 12:00 PM",12
```

JiffyDOS Examples:

```
@"$=T",12
@"$=T2"
@$=T2:*=P
@$=T4:*=S,N,>12/21/18 12:01 AM,<12/31/18 12:00 PM
```

Command Reference

Subdirectory commands are compatible with the syntax used by the CMD drives, although drive/partition numbers are ignored.

TODO: Test this, "drive/partition numbers are ignored". Confirm, and add additional commentary.

Creating Subdirectories

The *Make Directory* command allows you to create subdirectories. This command allows you to use standard path syntax. Using a path allows you to create a subdirectory in any partition or directory regardless of what your current partition or current directory may be. Here is the syntax to make a subdirectory:

```
OPENlf,dv,15:PRINT#lf,"MD[n][path]:name":CLOSElf
```

Where:

Variable	Description
lf	the logical file number for the command channel
dv	the current device number of the SD2IEC
n	the partition number where the subdirectory is to be created
path	the path to the subdirectory in which the new subdirectory will be created
name	the name of the new subdirectory

The above syntax may seem slightly confusing. Here are some guidelines to help you understand how this syntax works:

1. The name of the subdirectory you wish to create must always be separated from the rest of the command by a colon (:).
2. If you are creating a subdirectory within another subdirectory, you must specify that subdirectory within the path unless it is your current directory.
3. If subdirectories are specified within the path of the command (to the left of the colon), each subdirectory name must fall between slash (/) characters (only 1 slash is needed between subdirectory names.)
4. Paths normally start at the current directory. If you want the path to start at the root directory (the main directory in that partition), the path

should begin with two slashes.
5. If the subdirectory is to be created in a partition other than the partition in which you are located, place the partition number at the start of the path (in front of any slashes).

The following examples should help clarify these guidelines:

```
OPEN15,12,15:PRINT#15,"MD:TEMP":CLOSE15
OPEN15,12,15:PRINT#15,"MD1:TEMP":CLOSE15
OPEN15,12,15:PRINT#15,"MD1//:TEMP":CLOSE15
OPEN15,12,15:PRINT#15,"MD1//TEMP/:TEMP2":CLOSE15
OPEN15,12,15:PRINT#15,"MD/TEMP/:TEMP2":CLOSE15
```

JiffyDOS Examples:

```
@MD:TEMP
@MD1:TEMP
@MD1//:TEMP
@"MD1//TEMP/:TEMP2"
@"MD/TEMP/:TEMP2",12
```

Moving Between Subdirectories

The *Change Directory* command allows you to move between subdirectories. This command employs the same syntax used in the *Make Directory* command. Using a path allows you to move to a subdirectory anywhere in the currently selected partition. This command will also allow you to change the currently selected directory in any other partition, but will not move you into that directory. In order to move to the current directory of a different partition, you must issue a *Change Partition* command. The *Change Directory* syntax is:

```
OPENlf,dv,15:PRINT#lf,"CD[n]{[←]|[[path][:]subname]}":CLOSElf
```

Where:

Variable	Description
lf	the logical file number for the command channel
dv	the current device number of the SD2IEC
n	the partition number where the subdirectory you wish to make the current directory exists
path	the subdirectory path leading to the subdirectory
subname	the name of the subdirectory

Note that you can include the back arrow immediately after *CD[n]* to move backwards one directory (to the parent). The back arrow cannot be combined with any subdirectory path information. See the examples below.

It is not required that you include the colon before the subdirectory name, as long as the subdirectory name is preceded by a slash. You can use wildcards anywhere in the path.

NOTE: CD is also used to mount/unmount disk image files. To change into a disk image the image file must be the last component in the path, either after the slash or colon character. From the root directory of a mounted disk image, use the back–one–directory command (*CD*←) to leave the disk image and unmount it. When you are in a mounted disk image, two slashes specify the root directory of the mounted image. (See: Disk Images)

Here are some examples of the *Change Directory* command:

```
OPEN15,12,15:PRINT#15,"CD:TEMP":CLOSE15
OPEN15,12,15:PRINT#15,"CD1//:TEMP":CLOSE15
```

```
OPEN15,12,15:PRINT#15,"CD1//TEMP/:TEMP2":CLOSE15
OPEN15,12,15:PRINT#15,"CD1←":CLOSE15
OPEN15,12,15:PRINT#15,"CD/TEMP/TEMP2":CLOSE15
```

JiffyDOS Examples:

```
@"CD:TEMP",12
@CD1//TEMP
@CD1//TEMP/TEMP2
@CD1←
@CD/TEMP/TEMP2
```

Deleting Subdirectories

The *Remove Directory* command allows you to delete a subdirectory. This command does not allow the use of paths in order to avoid problems with removing a subdirectory which is a parent of the directory in which you are located. This command will not allow you to delete a subdirectory which contains any files or subdirectories. You must delete these files first by using the Scratch command. The following syntax applies to the *Remove Directory* command:

```
OPENlf,dv,15:PRINT#lf,"RD[n]:subname":CLOSElf
```

Where:

Variable	Description
lf	the logical file number for the command channel
dv	the current device number of the SD2IEC
n	the partition number where the subdirectory you wish to remove

	exists
subname	the name of the subdirectory you wish to remove

Here are some examples of the *Remove Directory* command:

```
OPEN15,12,15:PRINT#15,"RD3:TEMP":CLOSE15
OPEN15,12,15:PRINT#15,"RD:TEMP2":CLOSE15
```

JiffyDOS Examples:

```
@"RD3:TEMP",12
@RD:TEMP2
```

Partitions Table of Contents

sd2iec supports multiple partitions on a single SD Card, similiar to that of the CMD drives. In the original Commodore DOS a drive number could be provided to select between drive mechanisms in dual disk drives. CMD's DOS maintained this concept, but extended it to specify partition numbers instead physical drive mechanisms. sd2iec supports this as well.

NOTE: partitions may be on separate physical media, depending on the specific SD2IEC hardware in use.

Partitions are numbered starting with 1, which is the default partition. After resetting the SD2IEC or changing media, sd2iec will automatically switch the current partition back to the default.

sd2iec supports a persistent partition stored in EEPROM on the device itself, independent of the SD media. See EEPROM file system below. Not all hardware configurations support the EEPROM partition. The following discussion only applies to hardware which makes it available.

To allow reliable access to the EEPROM file system no matter how many partitions are available on the current SD media, a special character has been introduced that will always access the EEPROM file system. When sd2iec sees a "!" character where it expects a partition number and the "!" character is directly followed by a colon (i.e. "!:"), it will access the EEPROM partition. Additionally "$!" will load the directory of the EEPROM partition, similar to "$1" to "$9" for partitions 1 to 9.

TODO: Test this, I was having issues loading the directory from this partition when it was not the default.

The EEPROM partition can still be accessed using the partition number assigned, but this number may change depending on the number of partitions on the current SD media.

EEPROM file system

WARNING: The EEPROM file system is a newly-implemented file system that may still contain bugs. Do not store data on it that you cannot afford to lose. Always make sure that you have a backup. Also, the format may change in later releases, so this partition may need to be reformated in the future.

SD2IEC devices always have an EEPROM to store the system configuration, but on some devices this EEPROM is much larger than required. To utilize the empty space on these devices (currently any microcontroller with at least 128K of flash), a special EEPROM file system has been implemented, which is exposed to the user as an another partition, albeit a small one. This can for example be used to store a small file browser or fastloader. This partition can be used independently of the removable storage media.

The EEPROM file system will always register itself as the last partition number. You can load a directory of partitions, with the command "$=P", to find the current partition number of the EEPROM file system or use the alias (!) to access it.

To simplify calculations, block numbers on the EEPROM file system are calculated using 256 bytes per block instead of the usual 254 bytes as used by Commodore drives. Internally, the allocation is even more fine-grained (using 64 byte sectors), which means that the number of free blocks shown on an empty file system may be less than the sum of the number of blocks of all files on a full file system.

The EEPROM file system does not support subdirectories. It can be formatted using the N: command as usual, but the disk name and ID are ignored. The capacity of the EEPROM file system varies between devices: On AVR devices it is 3.25 KBytes and at most 8 files can be stored on it. On a2iec, the file system can hold 7 KBytes and at most 16 files can be stored on it. The actual number of files that can be stored depends on the length of the files, longer files need more than one directory entry.

Despite the apparent limitations, with some creativity it can be very handy to have some small persistent storage available at all times.

Disk Images

When CMD designed their storage devices in the late 80s early 90s, they included support for *emulation mode* partitions. These are partitions designed to mimic the tracks and sectors as well as the BAM and directory layout of 1541, 1571 and 1581 disks. Other technical details, such as error codes, job queue codes and certain key memory locations were duplicated to try to maintain compatibility with existing software.

With the advent and rise in popularity of Commodore emulators, such as VICE, it has become very common for Commodore 64 and 128 software to be distributed in *Disk Image* format. A disk image is a sequential, byte–for–byte copy of the tracks and sectors of a disk or partition. Some disk images may optionally include additional information to indicate *bad sectors*.

Software exists to copy a disk image from a file into a corresponding emulation mode partition on a CMD device, however this is not a very convenient way to work with disk images. And the bad sectors feature is not supported.

sd2iec significantly improves upon this situation. Emulation partitions are not supported, however, disk images may be mounted in place. This is very convenient. Each disk image can become a de facto emulation partition without needing to first move or copy the data. Compatibility is still limited, but disk image mounting is a much more modern and convenient alternative to the emulation partition concept from the CMD devices.

Supported Disk Image Types

The following disk image types are supported:

Device	Extension	File Size
1541	.D64	174848
1541 (with bad sectors)	.D41	175531
1571	.D71	349696
1571 (with bad sectors)	.D71	351062
1581	.D81	819200
CMD Native Partition	.DNP	multiple of 65536

Whenever the documentation talks about D64 images, the text applies to all of the above disk image formats, unless specifically noted.

Disk images are recognized by their file extension. If the image has an error info block appended it will be used automatically to simulate read errors. Writing to a sector with an *error* will always work, but it will not clear the indicated error. D81 images with error info blocks are not supported.

WARNING: There is at least one program (DirMaster v2.1/Style by THE WIZ) which generates broken DNP files. The usual symptom is that moving from a subdirectory that was created with this program back to its parent directory using *CD←* sets the current directory to an incorrect sector. You should avoid creating subdirectories with this version of DirMaster. It is possible to fix this problem using a hex editor, but the procedure is beyond the scope of this document.

Changing Disk Images

Each disk image file represents one disk side (of a single sided disk). Many programs span more than one disk image. To change the currently mounted disk image from within a program that does not support sd2iec specifically, there are hardware NEXT and PREV buttons.

A swap list is required to change the mounted disk image using the NEXT or PREV hardware buttons.

TECHNICAL ASIDE: Not all SD2IEC devices include these physical buttons. If your hardware does not have them it may be possible to add them yourself. Each button connects a certain pin on the AVR (the main microcontroller chip) to ground. The following is an overview of the necessary connections. If your SD2IEC already includes NEXT and PREV buttons (or you are uninterested in having this feature) skip over this table.

Original MMC2IEC ("larsp")	The NEXT button is on input PA4, the PREV button is on PA5. PA4 is pin 36 on the DIL version of the controller or pin 33 on the surface-mount version. PA5 is pin 35 on DIL, pin 32 on SMD.
Shadowolf's MMC2IEC 1.x PCBs ("sw1")	The NEXT button is on input PC4, the PREV button is on PC3. PC4 is pin 26 on the DIL version of the controller or pin 23 on the surface-mount version. PC3 is pin 25 on DIL, pin 22 on SMD.
Shadowolf's SD2IEC 1.x PCBs ("sw2")	The two required pins are available on the pin header which runs parallel to the long side of the board. In the documentation of the board, the NEXT button is named "DISKSWITCH", the PREV button is named "RESERVE".
Any other circuit	Any other circuit without disk change pin on a convenient connector somewhere and no button dedicated to that function: Please check with the supplier of the board and read config.h in the sources to find out how to connect it.

Creating a swap list

You may create a swap list file manually or let sd2iec create one for you.

A swap list is a text file with one line per disk image or directory you want to be able to change into. You are not limited to using disk images, a swap list file may also refer to standard directories on the SD card or anything else the *CD* command will accept. Blank lines are ignored, and parsing is tolerant of varying line–endings. sd2iec should be able to parse the file whether you create it on a Windows, Mac or Unix system, or even on a C64 itself. See *Creating a swap list file in BASIC* below.

By default sd2iec assumes the swap file is encoded in ASCII. If the the first line of the file exactly reads "#PETSCII" (in hex: $23 $50 $45 $54 $53 $43 $49 $49), filenames are assumed to be encoded in PETSCII instead. The PETSCII marker line will be skipped while navigating the swap list.

An example swap list file could look like this:

```
FOO.D64
BAR.D64
BAZ.D64
```

Or,

```
//NEATGAME/:DISK1A.D64
//NEATGAME/:DISK1B.D64
//NEATGAME/:DISK2A.D64
//NEATGAME/:DISK2B.D64
```

Only one swap list may be enabled at a time. The current swap file is assigned by sending *XS:filename* over the command channel where *filename* is the name of the swap list file. A swap list file remains enabled until explicitly disabled by sending *XS* on the command channel or manually assigning a different swap list.

While no swap list is currently assigned, if either the NEXT or PREV button is pressed, sd2iec will automatically activate a swap list named *AUTOSWAP.LST* (if such a file is found.) Any swap list automatically assigned in this manner can be manually disabled with the *XS* command, but will also be automatically disabled as soon as the sd2iec gets a *CD* (change directory) command. This way a different AUTOSWAP.LST file is always correctly recognized after you have changed into a different directory.

sd2iec can even auto-generate a swap list for you. An auto–generated swap list will contain all disk images (D64, D41, D71, D81 and DNP) found in the current directory.

To automatically generate a swap list, change into the directory that you want scanned and use the *HOME* function. (See: Navigating a swap list below). sd2iec will create a file called AUTOSWAP.GEN and activate it as if it were the standard AUTOSWAP.LST, including its auto–deactivation feature. The AUTOSWAP.GEN file will not be recognized the same way as AUTOSWAP.LST. To reuse the AUTOSWAP.GEN file you will need to either rename it (R:AUTOSWAP.LST=AUTOSWAP.GEN) or have sd2iec to generate it again using the *HOME* function. Use of an alternative name was adopted to avoid accidentally overwriting a pre–existing AUTOSWAP.LST file. And to allow sd2iec to recognize newly-added disk images in the directory without manually removing the generated swap list.

NOTE: Auto–generating a swap list in a directory where no disk images were found will produce an empty AUTOSWAP.GEN file. This may change in a future update.

Creating a swap list file in BASIC

Creating a swap list file using BASIC is very straightforward. No text editor is required. Here is an example listing that will create (or overwrite if it already exists) a swap list file named *swaplist*, on device 8 in the current directory. Customize the PRINT#2 lines to specify the filenames to be added to the swap list file.

```
10 OPEN2,8,2,"@0:SWAPLIST,P,W"
20 PRINT#2,"DISK A.D64"
```

```
30 PRINT#2,"DISK B.D64"
40 PRINT#2,"DISK C.D64"
50 PRINT#2,"DISK D.D64"
60 CLOSE2
```

Save this program to be able to quickly recreate a modified version of the swap list file. After running the program, assign the file as the current swap list like this:

```
OPEN2,8,2,"XS:SWAPLIST":CLOSE2
```

Or with JiffyDOS:

```
@"XS:SWAPLIST"
```

Navigating a swap list

- Press the NEXT button to unmount any currently mounted disk image and automatically mount the next image, or change into the next directory, listed in the swap file.
- Press the PREV button to unmount any currently mounted disk image and automatically mount the previous image, or change into the previous directory, listed in the swap file.
- Press the NEXT and PREV buttons together to return *HOME*. Unmount any currently mounted disk image and automatically mount the first image, or change into the first directory, listed in the swap file. Creates an AUTOSWAP.GEN file if no swap list is currently assigned.

Pushing either NEXT or PREV buttons will wrap around to the other end if they reach the beginning or end of the swap list.

sd2iec confirms each navigation actions with a specific LED flashing pattern. All actions begin by flashing both red and green LEDs for a short moment. For the NEXT function, the green LED will then flash on its own. For the PREV function, the red LED will then flash on its own. Both LEDs will be flashed again for the HOME function.

If any of these three functions is activated without an active swap list and sd2iec finds an AUTOSWAP.LST file, they will all be treated as the HOME function: The first line of the file is active and the red and green LEDs both flash twice. The same happens when an AUTOSWAP.GEN file is created, although the flashing pattern may not be very discernible because of the preceding card activity.

Command Reference

Mounting a Disk Image

Only one disk image at a time may be mounted. To mount a disk image you use the *CD* (Change Directory) command to change into the disk image, as though it were a subdirectory. The disk image may be specified as the final part of a path. However, directories contained within the disk image may not be specified in the same path used to mount the disk image.

The following syntax may be used to mount a disk image.

```
OPENlf,dv,15:PRINT#lf,"CD[n]{[path]|[:]}di":CLOSElf
```

Where:

Variable	Description
lf	the logical file number for the command channel

dv	the current device number of the SD2IEC
n	the partition number where the disk image you wish to mount exists
path	the subdirectory path leading to the disk image
di	the name of the disk image

Examples:

```
OPEN15,12,15:PRINT#15,"CD:FUNGAME.D64":CLOSE15
OPEN15,12,15:PRINT#15,"CD1//UTILITIES.DNP":CLOSE15
```

JiffyDOS Examples:

```
@"CD:FUNGAME.D64",12
@CD1//UTILITIES.DNP
```

Unmounting a Disk Image

After previously mounting a disk image, it is necessary to unmount a disk image before navigating to a different partition or subdirectory of the SD Media. Unmounting is performed using the back-one-directory feature (*CD←*) of the Change Directory command, from the root directory of the disk image.

If you are currently in a mounted disk image that supports subdirectories (i.e. a disk image of a CMD native mode partition, .DNP), the *CD←* is also used to go back one directory within the disk image. *CD//*, which is used to return to the root directory, returns you to the root directory of the mounted disk image, not of the current partition on the SD Media. A reliable method to unmount a disk image, then, is to first issue the command *CD//* followed by *CD←*

The command syntax to unmount a disk image is as follows:

```
OPENlf,dv,15:PRINT#lf,"CD←":CLOSElf
```

Where:

Variable	Description
lf	the logical file number for the command channel
dv	the current device number of the SD2IEC

Example:

```
OPEN15,12,15:PRINT#15,"CD←":CLOSE15
```

JiffyDOS Examples:

```
@"CD←",12
@CD←
```

Partition Numbers in Filenames

A partition number may be specified within a filename in place of the drive number. The drive number is the number which can precede the colon (:) in Commodore DOS filenames. Often, this number is not included since it is normally used only with dual drives to determine if the file operation should be directed to drive 0 or 1. Whenever this drive number is left out of the command, drive 0 is assumed. These rules also apply to sd2iec, which instead of containing 1 or 2 drives, can contain numerous partitions on a single SD Card. Whenever you wish to perform a command or file operation with the current partition, you may use a 0 or leave out the partition number entirely. However, if your current partition is different than the partition in which you wish to perform the file operation, you must either move to that partition first (with the *CP* command), or specify the partition number within the filename. The following examples should help illustrate this:

```
LOAD"1:MCOPY",12
OPEN2,12,2,"3:TESTFILE,S,W"
```

Load commands may be abbreviated with JiffyDOS:

```
/1:MCOPY
```

Partition Numbers in Disk Commands

As described above, partition numbers may be used to replace the drive number in filenames. Partition numbers may also be used in this manner when sending disk commands. In fact, any disk command which allows inclusion of a drive number (with the exception of direct access commands), will also allow you to substitute a partition number in its place. You may use partition numbers when copying, renaming and scratching. This is a large part of what makes sd2iec so compatible with CMD and Commodore DOS.

In the case of direct access commands, the current partition is assigned to the direct access file when that file is opened. Once the file has been opened, all commands sent to that file will refer to the partition you were in when the file was opened, even if the current partition is changed.

Creating and Deleting Partitions

sd2iec does not support creating or deleting partitions on the SD Media. SD Media typically ships ready to use, pre–formatted with a single partition. A PC, Mac or other computer must be used to repartition the SD Media.

NOTE: While the CMD HD supports up to 255 partitions, SD2IEC can only access 16 partitions. Two of those 16 are used automatically by the SYSTEM partition (0) and the EEPROMFS (the last partition), leaving SD2IEC with access to only 14 partitions on the SD Media. An SD Card can be configured with more than 16 partitions, but its partitions 15 and up are simply inaccessible.

OTHER PLATFORM NOTE: PC/Mac storage devices require a partition scheme to support partitions. For compatibility reasons, SD Cards ship with a Master Boot Record partition scheme. Master Boot Record (MBR) is an older standard, dating back to 1983. Apple Macintosh computers used a different scheme called Apple Partition Map, (APM). Apple Partition Map is not supported by sd2iec. A newer partition scheme, now used by both Windows and macOS, is called GUID Partition Table, (GPT). GPT supports much larger storage devices, however, it is not supported by sd2iec.

If you intend to format or partition an SD Card, make sure to use the Master Boot Record partition scheme, and the FAT16 or FAT32 file system. On macOS, the easiest way to create a multi-partition SD Card is with the command line utility *diskutil*. Insert the SD Card and then list disk devices to find the SD Card's device *identifier*, with the following command:

```
diskutil list
```

The identifier is listed beneath a column on the far right, *IDENTIFIER*. Disk identifiers use the format: diskXsY where X is a device number, and Y is a volume or partition number. E.g. disk4 (the whole device), disk4s1 (the first partition on disk device 4), disk4s2 (the second partition on disk device 4).

WARNING: formatting or re–partitioning a device will destroy all of its data. Make sure to backup to another device any data you do not wish to lose before proceeding.

You can partition the SD Card with a single command. The syntax is as follows:

```
diskutil partitionDisk did np MBR FAT32 pn psz [FAT32 pn psz]...
```

Where:

Variable	Description
did	the disk device identifier (eg. disk3, or disk4)
np	the number of partitions to create
pn	the name of this partition (Uppercase letters and numbers only, max. 11 Characters)
psz	the size of this partition (percentage of total device capacity, or R for remaining)

If you specify the number of partitions as only 1, the size of the partition would usually be 100%. If you specify less than 100%, the remaining space will be left unusued. If you specify more than 1 partition, the final 3 parameters must be repeated once for each partition, the parameters are: File System, Name, Size. FAT32 can be specified as the File System for each partition, each partition can be assigned a unique name. (Uppercase Letters and numbers only, maximum of 11 characters, to ensure maximum compatibility.)

Partition sizes can be specified in percentages, this is very convenient. Two partitions can be created as 50% and 50%, or 25% and 75%, etc. The final partition can have its size specified as *R* and it will automatically get the remaining space.

See AppleGazette, pro terminal commands article for details on using diskutil from the macOS terminal.

Changing Partitions

You may change from one partition to another by sending the *CP* (Change Partition) command via the command channel. The syntax is:

```
OPENlf,dv,15:PRINT#lf,"CPn":CLOSElf
```

Where:

Variable	Description
lf	the logical file number for the command channel
dv	the current device number of the sd2iec
n	the partition number you wish to change to (1-254)

Example:

```
OPEN15,12,15:PRINT#15,"CP4":CLOSE15
```

JiffyDOS Example:

```
@CP11
```

The *C{SHIFT}P* command is a variation on the Change Partition command which allows it to be used more easily from within BASIC programs. A shifted *P* is indicated by a PETSCII graphic symbol, or by a capital P while other characters are lowercase in uppercase/lowercase mode. This command allows you to use a character string to indicate the partition. The syntax is:

```
OPENlf,dv,15:PRINT#lf,"C{SHIFT}P"+CHR$(n):CLOSElf
```

Where:

Variable	Description
lf	the logical file number for the command channel
dv	the current device number of the sd2iec
n	the partition number you wish to change to (1-254)

Example:

```
OPEN15,12,15:PRINT#15,"C{SHIFT}P"+CHR$(11):CLOSE15
```

Formatting Partitions

sd2iec does not support formatting partitions on the SD Media. SD Media typically ships ready to use, pre–formatted with the FAT file system. A PC, Mac or other computer must be used to reformat the SD Media. The Commodore DOS NEW command and the BASIC 7.0 HEADER command are supported, but only for formatting a mounted disk image.

The standard Commodore DOS NEW command (not to be confused with the BASIC NEW command) may be used to format a mounted disk image from either BASIC 2.0 or BASIC 7.0. Although sd2iec will also accept the BASIC 7.0 HEADER command, this command is limited to either the current partition (0) or partition 1.

The DOS NEW commands may be used to delete all files from a disk image. The NEW command only works when a disk image is currently mounted. (See: Disk Images.) This command will be ignored when a .DNP image is mounted, unless the current directory of the disk image is the root directory.

When using the DOS NEW command, sd2iec can accept both the long and short versions. The BASIC 2.0 or BASIC 7.0 syntax for the DOS NEW command is as follows:

TODO: Test if sd2iec supports one mounted disk image per partition.

```
OPENlf,dv,15:PRINT#lf,"N[n]:partname[,id]":CLOSElf
```

Where:

Variable	Description
lf	the logical file number for the command channel
dv	the current device number of the sd2iec
n	the partition number you wish to format (1-254)
partname	the name you wish to appear in the partition header
id	a two character id for the partition header

The following syntax applies to the BASIC 7.0 HEADER command:

```
HEADER"partname"[,Iid][,Dn][{ON|,}Udv]
```

Where:

Variable	Description
partname	the name you wish to appear in the partition header
id	a two character id for the partition header
n	the partition number you wish to format (0 or 1)
dv	the current device number of the sd2iec

Examples:

```
OPEN15,12,15:PRINT#15,"N3:PARTITION 3,P3":CLOSE15
HEADER"PARTITION 1",IP1,D1 ON U12
```

JiffyDOS examples for using the DOS NEW command:

```
@N3:PARTITION 3,P3
@"N3:PARTITION 3,P3",12
```

Partition Directory

Having multiple partitions on an SD Card necessitates having the ability to view a directory of partitions. The partition directory may be viewed while you are working within any partition and relates information concerning the number, name and type of each partition on the SD Card. This command also allows you to filter partitions by pattern matching on the name. The syntax for this command is as follows:

```
LOAD"$=P[:pattern]",dv
```

Where:

Variable	Description

pattern	pattern to match on partition names
dv	the current device number of the sd2iec

All partitions are listed as type FAT. This may change to NAT in a future update, to improve compatibility

Examples:

```
LOAD"$=P",12
LOAD"$=P:S*",12
LOAD"$=P:?A*",12
```

JiffyDOS Examples:

```
@$=P
@"$=P:S*",12
```

Renaming Partitions

TODO: Test if sd2iec supports this. I don't think it does.

Renaming Partition Headers

TODO: Test if sd2iec supports this. What about in disk images?

Getting Partition Information

CMD DOS added the "Get Partition Info" command for the purpose of gathering information about the current or some other partition. The command is useful to the programmer whose software must react differently to partitions of various type and size. The partition number may be placed into a variable and inserted into the command as a character string. The syntax for the G-P command is:

```
OPENlf,dv,15:PRINT#lf,"G-P"[+CHR$(n)]:CLOSElf
```

Where:

Variable	Description
lf	the logical file number for the command channel
dv	the current device number of the sd2iec
n	the partition number (0-255)

If the *G-P* command is sent without the optional character string or a value of 255, the information returned will be for the current partition. A value of 0 requests that the information be returned for the system partition. Thirty bytes (0-29) of information concerning the requested partition plus a CHR$(13) are returned over the drive error channel.

sd2iec does not fully support this command, because it is designed to provide information about CMD native mode and Commodore device emulation partitions. The reported information is, therefore, partially faked. This command is supported for backwards compatibility purposes only. Feedback is welcome. Refer to the CMD Hard Drive Users Manual, Section 9-16 for full details.

Device Management Table of Contents

sd2iec supports a number of features for working with an SD2IEC hardware device itself. These features include, sleep mode, device detection, SD Card detection, memory accesses, rebooting, and updating firmware. For information about its software configuration and additional settings, see Settings.

Updating the sd2iec Firmware

One feature of the sd2iec firmware is its ability to automatically update itself with new firmware that is found on the storage medium.

To update the firmware on your SD2IEC, download *sd2iec-current-binaries.zip* from https://sd2iec.de. Unzip the package to find multiple versions of the firmware, compiled for different SD2IEC hardware devices.

It is not necessary to know which hardware device you have, nor which firmware version you currently have installed. Place all of the firmware binaries in the root directory of an SD Card. Power off your Commodore computer and your SD2IEC device, if it is powered independently from the computer. Insert the SD Card containing the new firmware binaries. Power on the Commodore computer and the SD2IEC device.

The bootloader will automatically search for firmware updates, detect them, and automatically install the latest version. It will automatically select the version which is compatible with the specific SD2IEC hardware it is installing to.

After new firmware is installed, you can confirm the current firmware version by performing the device detection procedure, see Device Detection.

Hot Swapping SD Cards

SD Cards may safely be removed and inserted while the computer is powered on. When a new SD Card is inserted, the current partition is reset to 1, and the current directory of all partitions is reset to the root directory.

If your hardware supports more than one SD card, changing either one will reset the current partition to 1 and the current directory of all partitions to the root directory. Doing so only for the card that would changed would be problematic if the number of partitions on the previous and the newly inserted cards is different.

Card Detection Test

Some SD slots suffer from bad or unreliable card detection switches. A diagnostic mode has been implemented to test for this condition.

Hold the PREV button during powerup. The red (dirty) LED indicates card detection status. The red LED will be lit if the card detection switch is closed. This does not indicate successful communication with the card, only that the mechanical switch on the SD card slot is closed.

On SD2IEC devices with two LEDs, the green (busy) LED indicates the state of the SD Card's write protect switch. The green LED will be lit if write protection is enabled. Due to the way an SD Card's write protect switch works, the indication is only valid when the card is fully inserted.

To exit from the diagnostic mode, power–cycle the SD2IEC or push the NEXT button once.

Sleep Mode

Some software uses fastloading routines that fail unless they are communicating with the only device on the IEC bus. The most recent notable example is the wonderful Sam's Journey. Sleep mode allows you to keep sd2iec connected to the serial bus even while load something from a different drive that uses this kind of fastloader. While in sleep mode SD2IEC does not listen to the bus at all.

To enter sleep mode, press and hold the SD2IEC's NEXT button for two seconds. Press and hold the NEXT button for two seconds again to wake up, rejoin the bus, and resume normal operation.

While in sleep mode, the red LED is on and the green LED is off.

Device Detection

Some programs need to identify the types of devices on the serial bus to provide special support for their features. It was common for older software to perform a *memory read* from the drive's internal memory, and the check for certain known bytes that were in a drive's ROM. This is a bad practice as it depends on the internal implementation of the drive's ROM. The detection can easily fail if the drive's ROM is updated.

sd2iec implements the M-R (Memory Read) command for backwards compatibility reasons only, but it should not be used by modern software. If you are the author of a program that needs to detect sd2iec for some reason, DO NOT use M-R for this purpose. Use the *UI* command instead and check the status message for specific device types or firmware versions, such as "sd2iec" or "uiec".

See, Warm, Cold and Hard Reset for information about the *UI* command.

Memory Access

Commodore disk drives, and the CMD family of storage devices, are effectively just trimmed down 8–bit microcomputers. They are usually based on the 6502 microprocessor, contain anywhere from 2 to 64 kilobytes of memory, as well as the DOS, file system and disk mechanism controller software in ROM. Commodore's DOS provides three memory related commands: M-R, M-W, and M-E, for memory read, write and execute.

Memory read takes a 16-bit starting address and optionally a number of bytes to read. If the optional number of bytes is omitted it defaults to one byte. The bytes are read directly from the drive's internal memory, either RAM or ROM, and are returned in the error channel (15). As mentioned in Device Detection, above, memory read was often used to identify the type of device, by looking for expected values at known locations.

Memory write takes a 16-bit starting address, the number of bytes to write, followed immediately by that number of bytes of data. Each call of the M-W command is limited to pushing 34 bytes of data into the device's memory. There are two common uses for a memory write. The first is to change device settings by poking new values into its workspace memory. This could be used to change the soft device number of a 1541, for example. This was a recommended practice, the address to write to is documented in Appendix A of the 1541–II User's Guide.

The second use is to push executable code into an area of free memory. This was also a recommended practice, as the DOS provides special *User Commands*, (U3, U4, U5, U6, U7 and U8) to begin executing code at one of 6 pre-defined addresses suitable for installing a small jump table. In addition to these user commands, the memory execute command can be used to instruct the drive to begin executing code at a provided address. If Memory Read is the equivalent of PEEK, and Memory Write is the equivalent of POKE, then Memory Execute is like the equivalent of SYS, only for the drive's memory and CPU instead of the computer's.

Executing code on a Commodore or CMD device works by assuming that the device is based on a 6502 microprocessor. Anyone familiar with writing programs in assembly language for the computer itself could translate that knowledge into writing programs to execute on the drive's own CPU. One of the most common uses for the drive to execute custom software is to support special fastloading routines.

sd2iec does not run on a 6502, but on an Atmel AVR microcontroller. In modern times it is considered bad practice for a device to expose its private or internal implementation. Doing so usually results in unnecessarily close coupling that results in future compatibility problems. For this reason, all memory access commands are emulated, and are provided only for legacy compatiblity purposes. sd2iec has specific ways of dealing with each command, and tries to handle each of the most common use cases from the past.

Memory Read (M-R)

By default, a small table of address–to–value pairs is built into the firmware. If a memory read attempts to read a byte from an address in this table, the corresponding value will be returned. The table contains the most commonly checked memory addresses for drive detection, and returns values designed to identify the SD2IEC as a 1541. If the address being read is not found in the table, more–or–less random data will be returned.

Some software (GEOS in particular) reads large portions of the drive's ROM as part of its drive detection. This amount of data cannot reasonably be stored in the microcontroller's firmware. sd2iec provides a solution to this problem. Using the *XR* command, (See: Settings), a ROM image file may be specified by name. The ROM image file is virtually mapped to the address range $8000-$FFFF. While a ROM image file is configured, memory reads to that address range are read in and passed through from the file.

A ROM image file should be a copy of the ROM contents of a 1541, 1571 or 1581 disk drive. Headers are skipped automatically. The name of the file must be 16 characters or fewer. When an M-R command is received sd2iec searches for the image file in three locations on the storage medium:

1. The current directory of the current partition

2. The root directory of the current partition
3. The root directory of the first partition

NOTE: The internal table will be used if the ROM image file cannot be found or an error occurs reading it. The ROM image file is used *only* with the M-R command.

Except for some very specific situations where drive detection fails (e.g. GEOS) using a ROM image will probably decrease compatibility with sd2iec. Most of the implemented fastloaders will only recognize the 1541 variation of the loader.

Memory Write (M-W) and Memory Execute (M-E)

Memory writes are entirely virtualized. Common addresses used to change internal settings of the 1541 are interpreted and used to change an analagous setting in sd2iec. For example, a memory write to the address where a 1541 stores its device number is intercepted and the value is used to change the device number of the sd2iec.

sd2iec also checks if the transmitted data in a memory write corresponds to any of a set of known software fastloaders. When a known fastloader is attempted to be written to memory, sd2iec internally selects its own implementation of that fastload protocol, which it will execute when the corresponding and expected memory execute command is called. See Fastloaders section below for more details on supported fastloaders. All other memory execute commands are ignored.

Command Reference

Device Address

The SD2IEC device address may be changed with the *U0* command, with the following syntax.

```
OPENlf,dv,15:PRINT#lf,"U0>"+CHR$(d):CLOSElf
```

Where:

Variable	Description
lf	a logical file number
dv	the current device number of sd2iec
d	the device number (8 to 30) to change to

Example:

```
OPEN15,8,15:PRINT#15,"U0>"+CHR$(12):CLOSE15
```

JiffyDOS Examples:

```
@"U0>"+CHR$(16)
```

The new device number is temporary, until the power is cycled or the sd2iec is reset. Changes to the device number are saved permanently when settings are saved. (See: Settings)

Other U0 commands are currently not implemented.

Warm, Cold and Hard Reset

The following commands can be sent via the command channel.

```
UI
```

Warm reset. Issuing UI just sets the "73,..." message on the error channel.

```
UJ
```

Cold reset. Issuing UJ closes all active buffers. However, it does not reset the current directory, mounted images, swap list or anything else.

```
U{SHIFT}J
```

Hard reset. This command causes the AVR processor to restart. If the bootloader is installed it will be skipped. ({SHIFT}J is equivalent to CHR$(202).)

Bus Protocol

The VIC-20 is able to access sd2iec using a faster bus protocol than the Commodore 64. The *UI-* command can be sent via the command channel to speed up the bus protocol, when used with a VIC-20 only. The VIC-20 is able to use sd2iec even when this command has not been issued.

Use the *UI+* command, via the command channel, to set the speed of the bus protocol back down for use on C64.

Switching to the slightly faster bus protocol for the VIC-20 works but has not been thoroughly tested.

Direct Access Table of Contents

The direct access commands on Commodore and CMD storage devices open the possibility for a wide range of custom file formats and data storage techniques by offering direct access to storage blocks. sd2iec offers similar support for Commodore and CMD disk images, and an extended direct access command set for the native FAT file system. To demonstrate the flexibility of direct access, it is helpful to understand how files are typically created.

Typical files are sequential chains of bytes. When a sequential file is created a first block is dynamically allocated and a directory entry is created that points to that block. As data are written to the file they are filled sequentially into the bytes from 2-255 of that block. The first two bytes, 0-1, are used to indicate if this block is the final block, and how much space in this block is used. When the current block is filled, a next block is allocated automatically and the first two bytes of the current block are converted into a pointer to the track and sector of the next block. The next block then becomes the current block. This process continues by allocating new blocks and chaining them to the end of the expanding file.

The direct access commands give the computer block–level read and write control over the storage medium. By reading and writing the block or blocks containing the Block Allocation Map (BAM), free blocks can be manually identified and allocated. Complex and custom non–sequential file formats can be created by devising entirely novel ways for blocks to be allocated, filled and pointed to within the file structure. These commands are designed to be used only by advanced programmers, as they allow you to organize data without help from the disk drive itself.

Buffers and Large Buffers

To use direct access you need two channels open to the device. One is the command channel (15), and another for transferring data. When opening the data channel, rather than specifying a file by name, you use the pound symbol (#) to specify opening access to a memory buffer. The memory buffer resides in the storage device.

A block read command can be sent over the command channel to specify a block to read from the storage medium into the memory buffer. The buffer pointer command can be used to set the position of the read/write pointer into that buffer. Reading a byte over the data channel retrieves one byte from the buffer and advances the buffer pointer by one. Writing a byte to the data channel puts one byte into the buffer and then advances the buffer pointer by one. A block write command can be used to write the contents of the memory buffer to any sector on the storage medium.

On Commodore and CMD storage devices, every sector is 256 bytes, and the memory buffer is also always 256 bytes. To support larger sector sizes of the FAT file system, sd2iec adds support for larger buffers.

A large buffer can be allocated by opening a file with two pounds symbols (##) followed by a single digit representing the size of the buffer in 256–byte increments. e.g. 2 for 512 bytes, 4 for 1024 bytes, etc. The syntax for opening a buffer is as follows:

```
OPENlf,dv,sa:PRINT#lf,"#[#sz]"
```

Where:

Variable	Description
lf	the logical file number (1 to 127)
dv	the current device number of the sd2iec
sa	the secondary address, or channel number (2 to 14)

sz	the size of a large buffer in multiples of 256 bytes

Examples:

```
OPEN4,9,4:PRINT#4,"#"
OPEN8,12,8:PRINT#8,"##2"
```

NOTE: When a standard buffer is opened the buffer pointer is set to 1 by default. However, when a large buffer is opened the buffer pointer is set to 0 by default.

If there aren't enough free consecutive 256–byte buffers to support the size requested a 70,NO CHANNEL message is returned on the error channel and no file is opened. For compatibility reasons, when attempting to open a large buffer, if the *filename* (i.e. ##x) isn't exactly three bytes long, a standard buffer ("#") will be allocated instead.

Reading and Writing Data with Direct Access

The block read (B-R or U1) and block write (B-W or U2) commands on sd2iec are only supported when a disk image is currently mounted. When B-R or B-W commands are being used, information is normally read or written using the BASIC commands GET#, INPUT# and PRINT#. The data being accessed with these commands is not being written to or read from the storage media directly, but is actually stored in a buffer which temporarily holds the data for a particular block on disk. This may seem a little confusing to those who have never used these commands before, so here is a simpler way of looking at the process.

Reading Data from the Drive

If you wish to read data directly from the disk image, first a B-R or U1 command is sent to sd2iec to tell it to read a particular block from the image. This block is placed into a RAM buffer in the SD2IEC. To transfer this data from the SD2IEC to the computer, the BASIC commands GET# or INPUT# are used. If you only need a portion of the data in the block, you can use the buffer pointer (B-P) command to tell the drive which byte will be the first to be passed to the computer with GET# or INPUT#.

Here is a quick example of reading the seventh byte from track 1 sector 0 of device number 12. The buffer points to byte 6 because the byte numbers start at 0, so byte 6 is actually the seventh byte.

```
10 OPEN15,12,15:         REM OPEN COMMAND CHANNEL
20 OPEN2,12,2,"#":       REM DIRECT ACCESS CHANNEL

30 PRINT#15,"U1";2;0;1;0:REM CHANNEL 2, TRACK 1, SECTOR 0
40 PRINT#15,"B-P";2;6:   REM CHANNEL 2, 7TH BYTE (6)
50 GET#2,A$:             REM READ THE BYTE INTO A$

60 CLOSE2:               REM CLOSE DIRECT ACCESS CHANNEL
70 CLOSE15:              REM CLOSE COMMAND CHANNEL
```

On a Commodore or CMD device, the actual buffer in memory can be specified. Then, knowing where in memory the buffer resides, you could also use a memory read (M-R) command to read data from the buffer. This is not supported by sd2iec. The memory read command is virtualized, and exists for limited legacy compatibility only. See Memory Access in Device Management.

Writing Data to the Drive

If you wish to write data directly to the disk image, the PRINT# command is first used to write this data to the memory buffer in the SD2IEC. After the data has been written to this buffer, the buffer itself is then transferred to the desired track and sector in the disk image by using the block write (B-W or U2) command.

Here is a quick example of writing to the seventh byte on track 1 sector 0 of device 12. As in the previous example, the buffer points to byte 6 because the byte numbers start at 0, so byte 6 is actually the seventh byte.

```
10 OPEN15,12,15:         REM OPEN COMMAND CHANNEL
20 OPEN2,12,2,"#":       REM DIRECT ACCESS CHANNEL

30 PRINT#15,"B-P";2;6:   REM CHANNEL 2, 7TH BYTE (6)
50 PRINT#2,CHR$(65):     REM WRITE BYTE TO BUFFER
40 PRINT#15,"U2";2;0;1;0:REM CHANNEL 2, TRACK 1, SECTOR 0

60 CLOSE2:               REM CLOSE DIRECT ACCESS CHANNEL
70 CLOSE15:              REM CLOSE COMMAND CHANNEL
```

The above example writes a sector to the disk image with no regard for what existed in that sector previously. If you want to change a byte in a sector without changing the rest of the contents of that sector, read it into the buffer first, change the desired byte to the new value, and then write the modified buffer back to the sector in the disk image.

Direct Sector Access

The block read and block write commands and command syntax are supported for legacy compatibility reasons and are valid only when a disk image is mounted. Those commands refer to 8-bit track and sector numbers which were valid for floppy disks and supported by CMD Native partitions allowing maximum addressable storage up to 16 megabytes.

The FAT file system supported by sd2iec organizes data in a partition in a continuous range of sequentially numbered clusters. Clusters are opaquely mapped to the underlying storage device's cylinders, tracks and sectors. Cluster numbers are 28–bit (32-bit numbers are used, but 4 bits are reserved for future use), for a maximum of 268,435,456 clusters per partition. Additionally, cluster size can vary from 512 bytes to 8KB, 16KB or more. See Creating and Deleting Partitions section in Partitions.

To accommodate direct access to the FAT file system's storage, sd2iec introduces a new command group for direct sector access. Some Commodore drives use D for disk duplication between two drives in the same unit. An attempt to use that command with sd2iec should result in an error message.

Direct sector access has three subcommands: Direct Info (DI), Direct Read (DR) and Direct Write (DW). Each of these commands require a buffer to be open, similiar to the B-R, B-W, U1 and U2 commands. Due to the larger sector size of the FAT file system, the direct read (DR) and direct write (DW) commands require a large buffer of size 2 (512 bytes) or larger. The direct info (DI) command, with page set to 0, fits in a standard 256 byte buffer.

If you try to use a command with a buffer that is too small a new error message is returned, "78,BUFFER TOO SMALL,00,00".

Command Reference

Allocating Blocks TODO: Test for support for Block Allocate (B-A). It is supported by the 1541.

Freeing Blocks TODO: Test for support for Block Free (B-F). It is supported by the 1541.

The Buffer Pointer

The buffer pointer (B-P) command allows you to point to a specific byte within a buffer which is to be read from or written to. This action occurs within the disk buffer which holds the data that was read with the block read command or is to be written with the block write command. The buffer in Commodore and CMD drives have a maximum size of 256–bytes, and so the pointer is an 8–bit number. Because sd2iec has support for large buffers, it adds an optional second byte allowing you to specify a 16-bit buffer pointer. If the second byte is not specified it defaults to zero.

The syntax for this command is as follows:

```
PRINT#lf,"B–P";ch;pl[;ph]
```

Where:

Variable	Description
lf	the logical file number used for the command channel
ch	the channel number used for the direct access file
pl	the 16-bit buffer pointer low address byte
ph	the 16-bit buffer pointer high address byte (defaults to 0)

The syntax given above assumes that the command and direct access channels have already been opened. The channel number is the same as the secondary address you used when opening the direct access chanel.

Reading Blocks

The block read (B-R) command has some quirks that make its behavior unexpected. The syntax is identical to the U1 command which behaves in a more consistent and expected way. Therefore, it is recommended to use U1 instead of B-R.

NOTE: The problem with the B-R command is that it will not read an entire block as data. Instead, it uses the first byte in the block as a counter for how many bytes are to be read. Therefore the full block will only be read if the first byte of the block contains a value of 255. Since the first byte of most blocks is a link pointer to the next sector, it is only valid to read blocks with B-R that were originally written with this first–byte–as—data-length schema, such as with the B-W command. To always read the full block, simply use the U1 command instead.

The syntax is as follows: (The same syntax applies for B-R, with the above–noted behavioral caveats.)

```
PRINT#lf,"U1";ch;n;t;s
```

Where:

Variable	Description
lf	the logical file number used for the command channel
ch	the channel number used for the direct access file
n	partition number (always 0)
t	the track from which to read the block
s	the sector to read into the buffer

B-R and U1 are only supported while a D64 image is mounted.

TODO: Test and confirm that this works with disk image types other than D64.

The channel number specified above is the same as the secondary address used to open the direct access file.

The partition number should always be 0 (zero). This is because direct access commands will always access the partition that was the current partition at the time the direct access channel was opened.

TODO: Test if B-R works after unmounting a disk image that was mounted when the direct access file was opened.

Writing Blocks

The block write (B-W) command has some quirks that make its behavior unexpected. It was designed to be paired closely with the B-R command. Due to this special behavior it is rarely used. Instead, most applications use the U2 command, which has the same syntax but behaves in a more consistent and expected way. Therefore, it is recommended to use U2 instead of B-W.

NOTE: The problem with the B-W command is that it will not write an entire block as data. Instead, it uses the first byte in the block as a counter for how many bytes are to be written. This will allow you to write (and later read with B-R) the full block only if the first byte of the block contains a value of 255.

The syntax is as follows: (The same syntax applies for B-W, with the above–noted behavioral caveats.)

```
PRINT#lf,"U2";ch;n;t;s
```

Where:

Variable	Description
lf	the logical file number used for the command channel

ch	the channel number used for the direct access file
n	partition number (always 0)
t	the track to which to write a block
s	the sector to write the buffer to

B-W and U2 are only supported while a D64 image is mounted.

TODO: Test and confirm that this works with disk image types other than D64.

The channel number specified above is the same as the secondary address used to open the direct access file.

The partition number should always be 0 (zero). This is because direct access commands will always access the partition that was the current partition at the time the direct access channel was opened.

TODO: Test if B-W works after unmounting a disk image that was mounted when the direct access file was opened.

Block Execute

Commodore and CMD devices are based on the 6502 microprocessor. Their DOS includes a block execute command which instructs the drive's processor to JSR to the start of the block and begin executing code that has previously been read into that memory buffer.

SD2IEC devices are not based on the 6502, and therefore sd2iec does not support the block execute command. See Memory Access section in Device Management.

Direct Info

Direct info is a command that allows you to get information about the current storage medium, for example the SD Card. The syntax for this command is as follows:

```
PRINT#lf,"DI"+CHR$(ch)+CHR$(dv)+CHR$(pg)
```

Where:

Variable	Description
lf	the logical file number used for the command channel
ch	the channel number used for the direct access file
dv	the number of the physical device to be queried
pg	the information page to be retrieved (currently always 0)

The only information page currently implemented is page 0. This may change in a future firmware update. After calling this command, an information block about the requested device will be loaded into the memory buffer, which can subsequently be accessed with BASIC GET# command over the data channel. The buffer pointer commmand can be used to specify which byte to retrieve first.

The structure of the information block is as follows:

Offset	Size	Description
0	1 byte	Number of valid bytes in this structure. This includes this byte and is meant to provide backwards compatibility if this structure is extended at a later time. New fields will always be added to the end so the offset and size of older fields will stay the same.
1	1 byte	Highest diskinfo page supported. Always 0 for now, will increase if more information pages are added. (Planned: Complete ATA IDENTIFY output for IDE, and CSD for SD)

2	1 byte	Disk type. This field identifies the device type, currently implemented values are: 0 IDE 2 SD 3 (reserved)
3	1 byte	Sector size in bytes, divided by 256. E.g. 2 = 512 bytes per sector, 4 = 1024 bytes per sector, etc.
4	4 byte	Number of sectors on the device, in little endian byte order. (Same byte order as the 6502.) If there is ever a need to increase the reported capacity beyond 2TB (for 512 byte sectors) this field will return 0 and a 64-bit value will be added to this diskinfo page.

If you want to determine if there is a device that responds to a given number, read info page 0 for it. If there is no device present that corresponds to the number you will see a DRIVE NOT READY error on the error channel and the "number of valid bytes" entry in the structure will be 0.

Do not assume that device numbers are stable between releases and do not assume that they are continuous either. To scan for all present devices you should query at least 0–7 for now, but this limit may increase in later releases.

Direct Read

Direct read is very similar to block read (U1) but with support for 32-bit sector references, and the ability to specify which device of multiple controlled devices. The syntax is as follows:

```
PRINT#lf,"DR"+chr$(ch)+chr$(dv)+chr$(sl)+chr$(sml)+chr$(smh)+chr
```

Where:

Variable	Description
lf	the logical file number used for the command channel
ch	the channel number used for the direct access file
dv	the number of the physical device to be queried
sl	the sector low byte (sector AND 255)
sml	the sector mid-low byte (sector/256 AND 255)
smh	the sector mid-high byte (sector/256/256 AND 255)
sh	the sector high byte (sector/256/256/256 AND 255)

BASIC on the C64 does not support positive integers larger than 32767 so it is not possible to represent the full 32–bit sector number in a single variable which is then processed by BASIC to send to sd2iec. However, the 32–bit sectors numbers can be tracked as 4 independent 8–bit values. The sector number is sent in little endian byte order.

Direct read (DR) reads the sector into the buffer. If an error occurs, the error channel will be updated with a 20,READ ERROR.

Direct Write

Direct write is very similar to block read (U1) but with support for 32-bit sector references, and the ability to specify which device of multiple controlled devices. The syntax is as follows:

```
PRINT#lf,"DW"+chr$(ch)+chr$(dv)+chr$(sl)+chr$(sml)+chr$(smh)+chr
```

Where:

Variable	Description

lf	the logical file number used for the command channel
ch	the channel number used for the direct access file
dv	the number of the physical device to be written to
sl	the sector low byte (sector AND 255)
sml	the sector mid-low byte (sector/256 AND 255)
smh	the sector mid-high byte (sector/256/256 AND 255)
sh	the sector high byte (sector/256/256/256 AND 255)

BASIC on the C64 does not support positive integers larger than 32767 so it is not possible to represent the full 32–bit sector number in a single variable which is then processed by BASIC to send to sd2iec. However, the 32–bit sectors numbers can be tracked as 4 independent 8–bit values. The sector number is sent in little endian byte order.

Direct write (DW) writes the buffer to the specified sector. If a write error occurs, the error channel will be updated with a 25,WRITE ERROR. If the sector failes to be written because the device is write protected, the error channel will be updated with 26,WRITE PROTECT ON.

Realtime Clock Table of Contents

Not all SD2IEC devices have a built–in realtime clock (RTC). However, if your hardware features an RTC, then the sd2iec time read and time write commands are available. If the RTC is not present, all time commands return a 30,SYNTAX ERROR,00,00 on the error channel. If the RTC is present but not set correctly time read commands will return 31,SYNTAX ERROR,00,00.

There are 4 types of commands provided for reading and writing the

SD2IEC's internal realtime clock. Each type of command uses a different format for sending and receiving clock data. The data types used are ASCII, ISO-8601, Decimal and BCD (binary coded decimal).

Reading Time and Date in ASCII Format

The T-RA command allows you to read the SD2IEC's clock and return the date and time as an ASCII string over the error channel. The syntax for this command is as follows:

```
OPENlf,dv,15:PRINT#lf,"T-RA"
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC

After the T-RA command is sent, the error channel will return the date and time in the following format:

```
"dow. mo/da/yr hr:mi:se xx"+CHR$(13)
```

Where:

Variable	Description
dow.	the day of the week. (4 characters followed by a space). SUN. MON. TUES WED. THUR

	FRI. SAT.
mo	the month (01-12)
da	the date (01-31)
yr	the year (year modulo 100, i.e. last two digits)
hr	the hour (01-12)
mi	the minute (00-59)
se	the second (00-59)
xx	either AM or PM

To read the error channel from BASIC, the following GET loop can be used:

```
10 GET#lf,A$:T$=T$+A$:IF ST<>64 THEN 10
```

Where *lf* is the logical file number that was used to open the error channel.

Writing Time and Date in ASCII Format

The T-WA command allows you to set the SD2IEC's internal realtime clock by sending an ASCII string representing the current time over the command channel. The syntax for this command is as follows:

```
OPENlf,dv,15:PRINT#lf,"T-WAdow. mo/da/yr hr:mi:se xx":CLOSElf
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC

The remaining parameters (*dow.,mo,da* etc.) follow the same format as described above under the T-RA command.

When the time is set, a year less than 80 is interpreted as 20xx.

NOTE: It is very important that the time and date parameters are separated by the same number of spaces and delimiters as shown above. There is no space between the "T-WA" and the first character of the day–of–week. Also, the day of the week *must* be four characters long and be followed by a space. (see T-RA for valid day-of-week strings.) If these parameters are not provided in the correct manner, the SD2IEC will not set the time correctly.

Reading Time and Date in ISO-8601 Format

CMD storage devices do not support this format. sd2iec adds support for a subset of this international date–time standard format. The T-RI command allows you to read the SD2IEC's clock and return the date and time as an ASCII string in ISO-8601 format over the error channel. The syntax for this command is as follows:

```
OPENlf,dv,15:PRINT#lf,"T-RI"
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC

After the T-RI command is sent, the error channel will return the date and time in the following format:

```
"YYYY-MM-DDThh:mm:ss dow"+CHR$(13)
```

Where:

Variable	Description
YYYY	the year (4-digits, i.e. 2019)
MM	the month (01-12)
DD	the date (01-31)
hh	the hour (24-hours, 00-23)
mm	the minute (00-59)
ss	the second (00-59)
dow	the day of the week. (3 characters). SUN MON TUE WED THU FRI SAT

This format complies with ISO 8601 and adds a day of week abbreviation.

Writing Time and Date in ISO-8601 Format

The T-WI command allows you to set the SD2IEC's internal realtime clock by sending an ASCII string representing the current time in ISO-8601 format over the command channel. The syntax for this command is as follows:

```
OPENlf,dv,15:PRINT#lf,"T-WIYYYY-MM-DDThh:mm:ss":CLOSElf
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC

The remaining parameters (*YYYY,MM,DD* etc.) follow the same format as described above under the T-RI command.

NOTE: It is very important that the time and date parameters are structured exactly as shown above. The year must include the century. There is no space between the "T-WI" and the first character of the year. Day of the week, if it is included with the T-WI command is ignored. Day of week is calculated automatically based on the date. Anything following the seconds field will be ignored.

ISO 8601 defines variations on this format, however, due to space constraints, sd2iec accepts only the format listed above.

Reading Time and Date in Decimal Format

The T-RD command allows you to read the SD2IEC's clock and return the date and time as a series of decimal-valued bytes over the error channel. This command provides BASIC (or Machine Language) programmers with a means to read the current time and date in numeric format from within a program. The syntax for the T-RD command is as follows:

```
OPENlf,dv,15:PRINT#lf,"T-RD"
```

Where:

Variable	Description
lf	the logical file number

dv	the current device number of SD2IEC

After the T-RD command is sent, the error channel will return the date and time as bytes in the following format:

Offset	Size	Description
0	1 byte	day of week (00=SUN.,01=MON., etc.)
1	1 byte	year (year - 1900 i.e. 108=2008)
2	1 byte	month (01-12)
3	1 byte	date (01-31)
4	1 byte	hour (01-12)
5	1 byte	minute (00-59)
6	1 byte	second (00-59)
7	1 byte	AM/PM flag (00=AM,non–zero=PM)
8	1 byte	CHR$(13)

Writing Time and Date in Decimal Format

The T-WD command allows you to set the SD2IEC's internal realtime clock by sending a series of decimal-valued bytes representing the current time over the command channel. The syntax for this command is as follows:

```
OPENlf,dv,15
PRINT#lf,"T–WD"+CHR$(BY0)+CHR$(BY1)+CHR$(BY2)+CHR$(BY3)+CHR$(BY4
CLOSElf
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC

BY0–BY7	the current time and date represented by eight decimal bytes (format of the bytes correspond with those given under the command T-RD, "Reading Time and Date in Decimal Format")

When the time is set a year less than 80 is interpreted as 20xx.

Reading Time and Date in BCD Format

The T-RB command allows you to read the SD2IEC's clock and return the date and time as a series of binary-coded-decimal bytes over the error channel. This format is similar to the decimal format but with a different byte encoding scheme. It can sometimes be more convenient to process BCD encoded data. The syntax is:

```
OPENlf,dv,15:PRINT#lf,"T-RB"
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC

After the T-RB command is sent, the error channel will return the date and time as BCD bytes in the following format:

Offset	Size	Description
0	1 byte	day of week ($00=SUN.,$01=MON., etc.)
1	1 byte	year (year modulo 100, i.e. 1990=$90)
2	1 byte	month ($01-$12)
3	1 byte	date ($01-$31)
4	1 byte	hour ($01-$12)

5	1 byte	minute ($00-$59)
6	1 byte	second ($00-$59)
7	1 byte	AM/PM flag ($00=AM,non–zero=PM)
8	1 byte	$0D

Writing Time and Date in BCD Format

The T-WB command allows you to set the SD2IEC's internal realtime clock by sending a series of BCD bytes representing the current time over the command channel. This command is normally sent from within a machine language program. The syntax for this command is as follows:

```
OPENlf,dv,15
PRINT#lf,"T–WB"+CHR$(BY0)+CHR$(BY1)+CHR$(BY2)+CHR$(BY3)+CHR$(BY4
CLOSElf
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC
BY0–BY7	the current time and date represented by eight decimal bytes (format of the bytes correspond with those given under the command T-RB, "Reading Time and Date in BCD Format")

When the time is set a year less than 80 is interpreted as 20xx.

Settings Table of Contents

sd2iec introduces a new family of extended DOS commands which are all prefixed with X. These commands are used to customize the behavior and settings of the SD2IEC device. After sending an extended command the setting is changed temporarily, until the device is reset (See: Warm, Cold and Hard Reset in Device Management), or the power is cycled. Temporary changes to settings can be made persistent by writing them to the EEPROM with the XW command.

Stored device settings can be bypassed during power up. Press and hold the disk change (NEXT) button during power up, and sd2iec will use the default settings instead of those stored in the EEPROM.

Command Reference

For all extended commands, if sending the command with JiffyDOS, the command string must be in quotes or you'll receive an error. The reason is because the X is interpreted as the JiffyDOS command to set the target device number for the built-in two drive file copier.

File Extension Mode

Commodore and CMD file systems contain a special byte in the directory entry for every file that specifies the type of the file, PRG, SEQ, USR, REL, etc. When a file is passed through non–Commodore compatible file systems this type byte is lost. Many stand–alone Commodore files, found on other platforms, simply have the file type code added as a 3–digit extension to the filename.

As an alternative to using a file extension, some Commodore emulators support a file wrapper around the original file. The wrapper contains the original 16-character filename, including characters that may be illegal in filenames on other file systems. The file extension mode command (XEx) lets you configure how sd2iec will create new files. Whether it should create files using a .XXX file extension or create a x00 file wraper.

See the Files section for further discussion of file extensions.

The syntax for this command is as follows:

```
OPENlf,dv,15:PRINT#lf,"XExm":CLOSElf
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC
xm	the extension mode number (0-4)

The following values are supported for extension mode:

Mode	Description
0	Never write x00 format files.
1	Write x00 format files for SEQ, USR and REL files, but not for PRG.
2	Always write x00 format files, for all file types.
3	Use file extensions for SEQ, USR and REL files, do not create a x00 header.
4	Use file extensions for SEQ, USR, REL and PRG files, do not create a x00 header.

The default mode is 1.

In mode 3 or 4, filename extension hiding is automatically enabled. The extension is interpreted instead as the Commodore file type. See Extension Hiding (XE+/-) below.

An exception exists for files created as type PRG and with a supported disk image extension (D64, D41, D71, D81, DNP, and M2I). For compatibility reasons, these files are never x00 wrapped and no .PRG filename extension is appended, as they are designed to be used on other platforms as is.

EDITOR'S NOTE: I have actually not been able to get these settings to work as described, with the latest firmware version on uIEC/SD. In my attempts, files are always wrapped in x00 wrappers, even when in mode 0.

Examples:

```
OPEN15,10,15:PRINT#15,"XE2":CLOSE15
OPEN10,8,15:PRINT#10,"XE0":CLOSE10
```

JiffyDOS Examples:

```
@"XE2"
@"XE0",12
```

Extension Hiding

Extension hiding is a close companion of the extension mode setting above. Extension hiding can be enabled or disabled. This setting determines how existing files will be interpreted. When enabled, files with a .prg/.PRG, .seq/.SEQ, .usr/.USR or .rel/.REL filename extension will have their extension removed and reinterpreted as the Commdore file type. E.g. "APPLICATION.PRG" (when viewed on a PC/Mac) will appear in the Commodore directory as a PRG type file named "APPLICATION". "readme.seq" will become a SEQ type file named "readme". When extensions hiding is disabled, filenames are presented as they are, and default to PRG type. (SEE: Long Filenames in Files.)

Existing x00 files are always implicitly *unwrapped*, regardless of file extension mode. For all commands the file is treated as having the filename that is in the x00 header. This is true for directory listings, loading, saving, pattern matching, copying, renaming, scratching, etc.

```
OPENlf,dv,15:PRINT#lf,"XE{+|-}":CLOSElf
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC
+ or -	+ is used to enable hiding. - is used to disable hiding.

Examples:

```
OPEN15,10,15:PRINT#15,"XE+":CLOSE15
OPEN10,8,15:PRINT#10,"XE-":CLOSE10
```

JiffyDOS Examples:

```
@"XE+"
@"XE-",12
```

Disk Image File Type

A disk image file is special on sd2iec because it can be mounted. Also, the command for mounting the image is the change directory command. This makes image files behave very much like (though not identical to) sub–directories.

The image mode command (XI) switches the display mode for mountable files. Either the file can be shown as a PRG type file, or it can be shown as a DIR type subdirectory. Even when listed as DIR it can still be accessed as a file. Lastly, an additional mode exists that will cause mountable files to appear in the directory listing twice. Once as a PRG file and once as a DIR subdirectory. Presenting the file as DIR type may help compatibility with software written for CMD devices. However, due to limitations in the CD command, such software may still fail to mount a disk image even with this option enabled. The syntax is as follows:

```
OPENlf,dv,15:PRINT#lf,"XItm":CLOSElf
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC
tm	the type mode (0-2) 0 = PRG 1 = DIR 2 = PRG and DIR

Examples:

```
OPEN15,10,15:PRINT#15,"XI0":CLOSE15
OPEN10,8,15:PRINT#10,"XI2":CLOSE10
```

JiffyDOS Examples:

```
@"XI0"
@"XI2",12
```

Modern Wildcard Pattern Matching

If modern wildcard pattern matching is enabled, characters after an asterisk (*) in a filename pattern are matched against the end of the filename. If disabled, any characters following an asterisk (*) are ignored.

The behavior of an asterisk matching all characters to the end of a filename was standard on Commodore disk drives prior to the 1581. The 1581's style of wildcard pattern matching more closely meets user expectations from modern computers but could break compatibility with older software. The syntax for this command is as follows:

```
OPENlf,dv,15:PRINT#lf,"X*{+|-}":CLOSElf
```

Where:

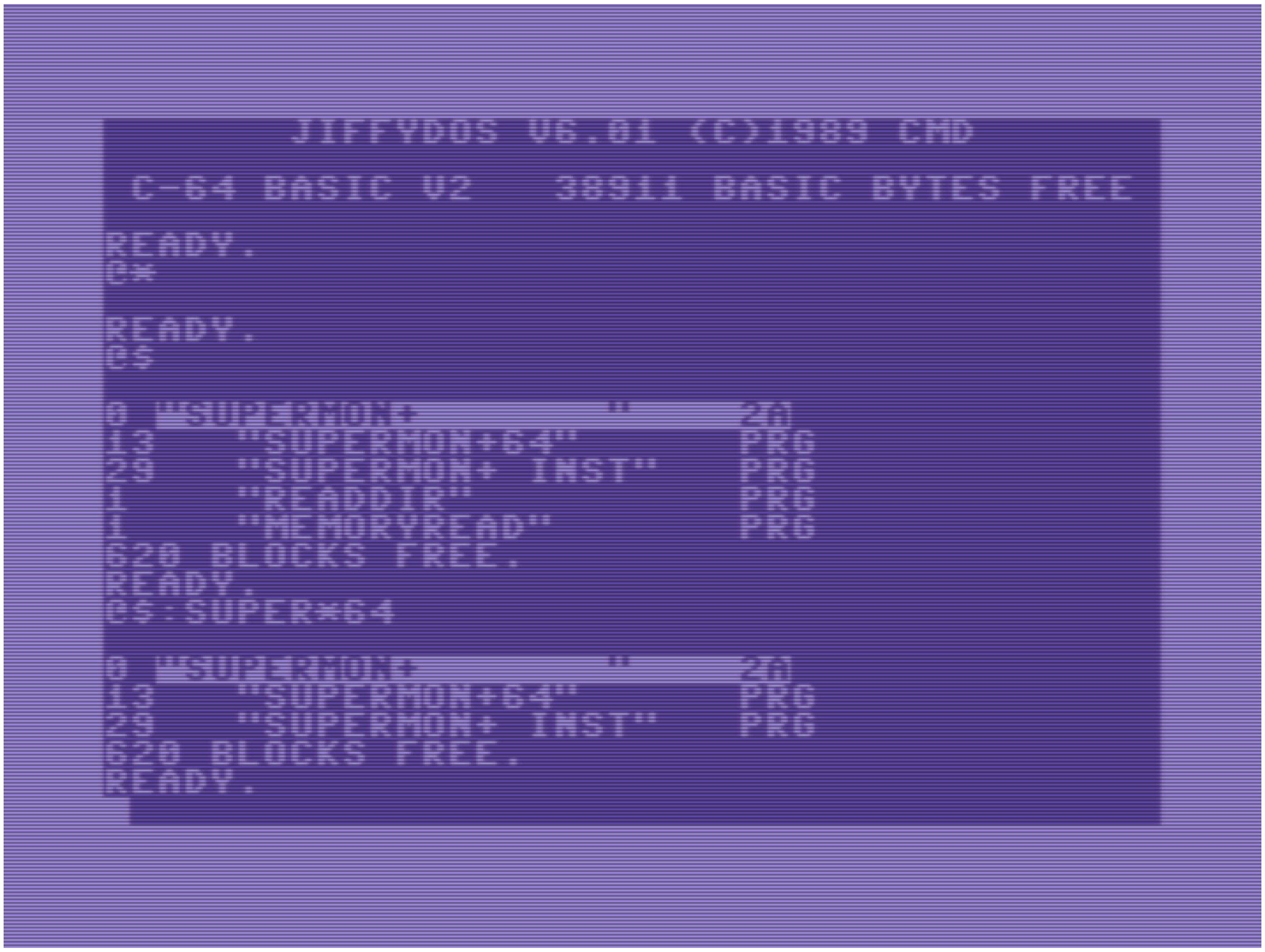
Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC
+ or -	+ enables modern wildcard pattern matching - disables modern wildcard pattern matching (defaults to enabled)

Examples:

```
OPEN15,10,15:PRINT#15,"X*+":CLOSE15
OPEN10,8,15:PRINT#10,"X*-":CLOSE10
```

JiffyDOS Examples:

```
@"X*+"
@"X*-",12
```



```
    JIFFYDOS V6.01 (C)1989 CMD
 C-64 BASIC V2   38911 BASIC BYTES FREE
READY.
@*

READY.
@$

0 "SUPERMON+           " 2A
13    "SUPERMON+64"       PRG
29    "SUPERMON+ INST"    PRG
1     "READDIR"           PRG
1     "MEMORYREAD"        PRG
620 BLOCKS FREE.
READY.
@$:SUPER*64

0 "SUPERMON+           " 2A
13    "SUPERMON+64"       PRG
29    "SUPERMON+ INST"    PRG
620 BLOCKS FREE.
READY.
```

Wildcard pattern matching on a 1541. Note the "64" following the "" is ignored.*

Configure Multiple Drives

Some SD2IEC hardware devices have support for multiple physical drives. On ATA–based units or units with multiple drive types, the *drives* command can be used to enable or reorder the drives.

Only devices supported by the specific hardware can be selected. Unsupported device types return an error if requested. You cannot configure one device in multiple drive slots. Finally, while it is possible to reorder ATA devices using this command, it is not recommended. Use the master/slave jumpers on the ATA devices instead.

To reset drive configuration, set all drive slots to "no device". The syntax for the drives command is as follows:

```
OPENlf,dv,15:PRINT#lf,"XDdn=dt":CLOSElf
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC
dn	the drive slot number to assign a drive to (0 to 7)
dt	the drive type to assign to the drive slot. 0 = Master ATA device 1 = Slave ATA device 4 = Primary SD/MMC device 5 = Secondary SD/MMC device 6 = (reserved) 15 = no device

Examples:

```
OPEN15,10,15:PRINT#15,"XD0=4":CLOSE15
OPEN10,8,15:PRINT#10,"XD1=0":CLOSE10
```

JiffyDOS Examples:

```
@"XD0=4"
@"XD1=15",12
```

View Drive Configuration

Viewing drive configuration is the companion of the drives command. This command is sent over the command channel (15), and the drive configuration information is returned in the error/status message. The syntax is as follows:

```
OPENlf,dv,15:PRINT#lf,"XD?":CLOSElf
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC

The status number is always returned as "03". The status message lists the drives in the following format:

```
D:dn=dt[:dn=dt...]
```

Where:

parameter	Description
dn	a configured drive slot number
dt	the drive type assigned to the drive slot

The track (the first value after the status message) indicates the current device address of SD2IEC. The sector (the second value after the status message) indicates extended drive configuration status information.

Example output:

```
03,D:00=01:01=04,10,01
```

Configure a Swap List

The swap list command allows you to specify a swap list file to use. (See: Changing Disk Images in Partitions) Here is the syntax for the swap list command:

```
OPENlf,dv,15:PRINT#lf,"XS:sl":CLOSElf
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC
sl	the name of the swap list file to enable

Example:

```
OPEN15,10,15:PRINT#15,"XS:MY SWAP LIST":CLOSE15
```

JiffyDOS Examples:

```
@"XS:MY SWAP LIST"
@"XS:A SWAP LIST",10
```

While the swap list is enabled, pushing the NEXT and PREV buttons on SD2IEC cycle the currently mounted disk image through the files listed in the swap list file.

There are several ways to disable the current swap list. You can issue the command again to enable a different swap list. The previous one will be disabled automatically. You can power cycle or push the reset button on the SD2IEC. You can issue the hard reset command (U{SHIFT}J). Or, you can issue the XS command without the colon (:) or filename.

EDITOR'S NOTE: In my experience, sd2iec is unable to find a swap list file by name if it is in fact wrapped in an x00 file wrapper. This can lead to confusion, as you may be able to see a file listed in the Commmodore directory, but when you try to assign that file as the swap list, if it is in fact wrapped in an x00, the command will only result in a FILE NOT FOUND error. (This is probably a bug in the firmware.)

Configure a ROM Image for Memory Read

The ROM image command allows you to specify a file for use with the memory read (M-R) command. (See: Memory Read in Device Management) Following is the syntax for the ROM image command:

```
OPENlf,dv,15:PRINT#lf,"XR:fn":CLOSElf
```

Where:

Variable	Description
lf	the logical file number
dv	the current device number of SD2IEC
fn	the filename for the ROM image

Example:

```
OPEN15,10,15:PRINT#15,"XR:1581.ROM":CLOSE15
```

JiffyDOS Examples:

```
@"XR:CMDHD-ROM.BIN"
@"XR:1571-DOS.ROM",10
```

While the ROM image file is mapped, the virtualized memory read commands to the address range $8000 to $FFFF are read from the image file. A file matching the assigned ROM image filename is searched for in the current folder, the root folder of the current partition, and finally the root folder of partition 1.

There are several ways to disable the ROM image file. You can issue the command again to enable a different ROM image file. The previous one will be disabled automatically. You can power cycle or push the reset button on the SD2IEC. You can issue the hard reset command (U{SHIFT}J). Or, you can issue the XR command without the colon (:) or filename.

Write Extended Settings to EEPROM

Changes to the sd2iec extended settings are changed only in memory. The changes only persist until the SD2IEC is power cycled or reset, or sd2iec is given a hard reset command.

When sd2iec is reset it reads its current settings from a persistent store of settings in the EEPROM. This command can be used to write the current in–memory settings to the EEPROM so that they will persist even after turning SD2IEC off.

Stored configuration includes:

- File Extension Mode
- File Extension Hiding
- Disk Image File Type
- Modern Wildcard Pattern Matching
- Multiple Drive Configuration
- ROM Image for Memory Read, and
- Current Device Address

If you change the device address by software, and write the change to EEPROM, sd2iec will power up with that address. Unless you also change the device address jumpers to a different setting than the one active at the time the configuration was saved. (NOTE: Not all SD2IEC hardware offers device address jumpers.)

The software device address command, in effect, assigns an address to the current configuration of the device address jumpers. This "hardware overrides software, overrides hardware" priority was chosen to allow accessing sd2iec even when it is soft-configured for a device number that is already taken by another device on the bus without having to remove that device to reconfigure sd2iec (e.g. when using a C128D).

The write extended settings command (XW) is sent to sd2iec over the command channel. Example syntax is as follows:

```
OPEN15,8,15:PRINT#15,"XW":CLOSE15
```

JiffyDOS Examples:

```
@"XW"
```

```
@"XW",12
```

View All Extended Settings

This command is used to list the state of all extended settings. Send this command to sd2iec on the command channel.

Example:

```
OPEN15,8,15,"X":CLOSE15
```

JiffyDOS Examples:

```
@"X"
@"X",12
```

The status byte is always 03. The status message lists the current state of all extended settings, similiar to DophinDOS. Each setting consists of the setting code followed immediately by its value. Settings are separated by colons (:). The track (first value after the status message) indicates the current device address.

Example output:

```
03,J-:C152:E01+:B+:*+,08,00
```

The view extended settings command can be appended with a question mark (?) to make an extended version query. The output of an extended version query includes the firmware version number, processor type, and the suffix of the configuration file which usually corresponds to the short name of the hardware sd2iec was compiled for.

Example:

```
OPEN15,8,15,"X?":CLOSE15
```

JiffyDOS Examples:

```
@"X?"
@"X?",12
```

Software Fastloaders Table of Contents

Many fastloading routines have been and continue to be used in both commercial and non–commercial Commodore 64 software. These routines often send program code to the disk drive, and then have the drive execute the uploaded code. These routines then work in concert with a custom bus access protocol in the software on the computer to accelerate the transfer of data.

SD2IEC devices are not built on a 6502 microprocessor, and are therefore unable to accept and run code in this manner. (See: Memory Access in Device Management). Without any additional support, all software that depends on these fastloader techniques would fail to load from sd2iec. While this situation would seem rather bleak from a compatibility standpoint, in practice the vast majority of software utilizes one of a small number of common fastloading techniques or protocols. sd2iec, therefore, attempts to detect which fastloading protocol a program is trying to use and activates its own custom–built emulation of that protocol. This is not a perfect solution, but it goes a long way towards improving compatibility with existing software, particularly when combined with disk images.

TECHNICAL NOTE: Using sd2iec without an external crystal or similiar precise clock source is not a supported configuration. Without a precise clock source, sd2iec will likely suffer from random data corruption. Some fastloader implementations actively refuse to work if they detect an unsuitable clock source.

Fastloader Support and Technical Notes

Action Replay 6

The Action Replay 6 reads a byte from the drive ROM to check which fastloader it should use. When file–based M-R emulation is disabled sd2iec returns a value that should force the cartridge to use the standard KERNAL loader instead of its many fastloaders/savers. This means that accessing sd2iec with a ROM image enabled will fail because the cartridge will enable a fastloader that will not be recognized.

Currently the only recognized Action Replay 6 fastloader and fastsaver are the ones for the 1581.

Dreamload

Dreamload uses direct track/sector access, so it is only supported when accessing a mounted disk image. As sd2iec has to constantly wait for commands from the C64 the NEXT and PREV buttons may become unresponsive. Try multiple times if necessary. Dreamload does not work with more than one device on the serial bus.

Dreamload is a captive fastloader. This means sd2iec stays in Dreamload mode until it receives a *quit loader* command from the C64. To force sd2iec to resume normal operation, hold the NEXT button until the red LED turns on (just like sleep mode).

ELoad Version 1

This loader was made for EasyProg but may also be used in other programs. It detects and supports the sd2iec natively.

Epyx FastLoad Cartridge

Only the fastloader from this cartridge is supported. The additional features of this cartridge, sector editor, file copier, file lock, etc. are not supported.

EXOS V3 and The Beast System

Both are supported. The loader used by these KERNALs is very similiar to the Final Cartridge III fastloader.

Final Cartridge III

Both the fastloader and the fast saver of Final Cartridge III are supported. The slightly different fastloader, used for files freezed with the Final Cartridge III, is also supported.

The Final Cartridge III is both PAL and NTSC compatible.

G.I. Joe Loader

The G.I. Joe loader is said to be the most-ripped IRQ loader. Unfortunately this is another *captive* fastloader (similiar to dreamload but not restricted to disk images because it is filename-based). There is no reliable way to detect if the computer has been reset to switch back to the standard protocol.

To exit this loader, hold down the SD2IEC's NEXT button until the red LED turns on (just like sleep mode).

JiffyDOS

JiffyDOS is supported by sd2iec natively.

The JiffyDOS protocol has very relaxed timing constraints compared to Turbodisk, but still not as relaxed as the standard Commodore IEC protocol.

Maniac Mansion

Original versions of Maniac Mansion have an additional copy protection check that is not supported by sd2iec. Please use a cracked version instead—the ones from Gamebase 64 seem to work. Remember to add an empty D64 for the save/load disk to your swap list if you want to save your game.

This game uses a captive loader that unfortunately cannot detect if it should exit automatically. To resume normal operation, hold down the SD2IEC's NEXT button until the red LED turns on (just like sleep mode).

Sam's Journey

The loader in Sam's Journey expects that there is only a single drive active on the serial bus and that this drive has ID 8. The reimplementation in sd2iec can handle both disk images and extracted files stored in a single directory directly on the SD card.

The loader is captive, but tries to exit cleanly when possible. If you reset the C64 while it is reading data (busy LED lit, e.g. during the intro), sd2iec will hang and requires a reset or powercycle to recover.

Turbodisk

Turbodisk is detected by the CRC of its 493 byte long floppy code and the M-E address 0x0303. The same code seems to be used under various names, among them "Turbodisk" (both 2.1 and 2.2) and "Fast-Load".

It is not known if there is an NTSC-compatible version of this fastloader.

ULoad Model 3

ULoad Model 3 uses direct track/sector access, so it is only supported when accessing a mounted disk image. Currently, there is only one supported variant of ULoad Model 3, which is the one used by Ultima 3 Gold. There are no other known variants at this time, but this may change.

If you are a coder and want to use ULoad Model 3 in your own program, either configure it to produce the same drive code as Ultimate 3 Gold, or contact Ingo Korb to work out a way to trigger ULoad Model 3 support without uploading any code to the drive.

Zak McKracken

Same as Maniac Mansion, but this game only has a code list protection, so images of original disks should work fine.

Operating System Support

C64 OS

C64 OS is sd2iec compatible. It does not employ any custom fastloading routines, but is JiffyDOS compatible. Thus, using JiffyDOS in conjunction with C64 OS is strongly recommended.

If sd2iec is used as the boot device of C64 OS, you cannot use the NEXT and PREV buttons to change disk images. Doing so would cause the boot volume to no longer be available on the bus.

GEOS

GEOS 2.0 can be booted from D64 images made from original disks as well as D41, D71 and D81 images created using geoMakeBoot. Make sure to *Configure* the system for a 1541, 1571, or 1581, respectively, before using geoMakeBoot.

When a ROM image is not in use, GEOS may detect sd2iec as either a 1541 or 1581, depending on the version of Configure used. This may cause the system to fail to boot. For example, if sd2iec is detected as a 1581 while booting from a D64 disk image. Use of a drive ROM image is recommended when using GEOS to avoid these problems.

GEOS 1.3 may or may not work. It boots, but has not received indepth testing. Gateway seems to work but has not been tested beyond booting from a D64 image.

Using the NEXT and PREV buttons for changing the current disk image is supported. However, make sure that you only access disk images that the drive type selected in GEOS would support (i.e. D64 for a 1541, D64 or D71 for a 1571 and D81 for a 1581).

Wheels

Wheels can be booted from any disk image type it supports. The correct ROM image file must be configured, especially for CMD HD emulation. (See: Memory Read in Device Management.)

Do not use the NEXT or PREV buttons to change disk images when booted from a disk image of a CMD Native Mode Partition (DNP) when the CMD HD ROM image is configured. Wheels does not check for disk changes on that type of drive. For other drive types, the same restrictions that GEOS has on disk image types also apply to Wheels.

Copyright and Credits Table of Contents

sd2iec - a controller/interface adapting storage devices to the CBM serial bus

The project was inspired by, and uses some code from, MMC2IEC by Lars Pontoppidan.

Parts based on code from others, see comments in main.c for details.
JiffyDOS send based on code by M.Kiesel
Fat LFN support and lots of other ideas+code by Jim Brain
Final Cartridge III fastloader support by Thomas Giesel
IEEE488 support by Nils Eilers

Deprecation notices

The following feature(s) will be removed in the next release:

M2I support

M2I support has been redundant since the introduction of transparent P00 support. To continue to use your M2I-format software, convert your files to P00 format (e.g. with m2itopc64.c) and set your device to extension mode 2 (XE2).

Do you like what you see?

You've just read one of my high–quality, long–form, weblog posts, for free! First, thank you for your interest, it makes producing this content feel worthwhile. I love to hear your input and feedback in the forums below. And I do my best to answer every question.

I'm creating C64 OS and documenting my progress along the way, to give something to you and contribute to the Commodore community. Please consider purchasing one of the items I am currently offering or making a small donation, to help me continue to bring you updates, in–depth technical discussions and programming reference. Your generous support is greatly appreciated.

Greg Naçu — C64OS.com

Want to support my hard work? Here's how!