Ultimate DOS

Command Summary

Gideon Zweijtzer

Version 1.1, September 12, 2013

Table of Contents

1. Introduction

1.1. Context

The “Ultimate DOS” provides a way to access the file-system of the 1541 Ultimate-II module programmatically.

“Ultimate DOS” is a target of the “Ultimate-II command interface”, and is thus accessible from the cartridge port, through some I/O registers. The document “Ultimate-II Command Interface – Register API” describes how commands are sent over this interface.

1.2. Purpose of this document

This document describes the commands that can be sent to this target, and their expected behavior and response.

2. Commands

In version 2.6 of the firmware of the 1541 Ultimate-II, the "Ultimate-DOS" target is accessible through target $01 and $02. This shall be the first byte of the command. Note that these two targets are *instances* of the DOS. They have their own state. This enables to have two directories and two files open at a given time; one on each target. In the examples in this chapter, target $01 is used.

The following paragraphs describe each of the commands of "Ultimate-DOS".

2.1. DOS_CMD_IDENTIFY (0x01)

Command format: `$01 $01`

The "Identify" command sends back an identification string, such as "`ULTIMATE-II DOS V1.0`". The user software can use this function to query which targets exist, or to obtain version information.

The status channel will report "`00,OK`", as this command cannot fail.

2.2. DOS_CMD_OPEN_FILE (0x02)

Command format: `$01 $02 [attrib] <filename>`

The "Open File" command takes two arguments: an attribute byte; directly followed by the filename to be opened. The attribute byte contains flags that tell the file system how, in which mode, to open the file. The following table shows which flags are applicable:

Attribute	**Value**
FA_READ	$01
FA_WRITE	$02
FA_CREATE_NEW	$04
FA_CREATE_ALWAYS	$08

To open a file in read mode, only just use $01. To open a file in write mode, use $02 if the file you write to already exists. This mode will not clear the file. *Add* $04 if you would like to create a new file to write; this mode will clear the file to 0 bytes first. *Add* $08 if the file you open may overwrite a file that already exists. (So, in order to open a file for writing that may always overwrite an existing file, use $0E.)

The filename does not need to be null-terminated, as the length of the command determines the length of the file name string.

Example: `$01 $02 $01 MYFILE`

The command will never return data. Status will either be "`00,OK`", or a status message from the file system.

2.3. DOS_CMD_CLOSE_FILE (0x03)

Command format: `$01 $03`

The "Close File" command closes the file that was last opened. It does not take any arguments, neither will this command return any data. The status channel will read:

"`00,OK`" or "`84,NO FILE TO CLOSE`"

2.4. DOS_CMD_READ_DATA (0x04)

Command format: `$01 $04 [len_lo] [len_hi]`

The "Read Data" command will start a read transfer from the opened file. If there is no file open, the reply will be an empty data packet, and the status channel will read "`85,NO FILE OPEN`".

The third and fourth bytes of the command indicate the total number of bytes that will be transferred by this command. The maximum is 65,535 bytes. However, data is always transferred in chunks of 512 bytes, max. The receiving software shall 'accept' the 512-byte data packet before more data is transferred. This is due to the maximum size of the message in the data queue.

When something goes wrong, this will be reported through the status channel. When everything is okay, the status channel will stay quiet.

2.5. DOS_CMD_WRITE_DATA (0x05)

Command format: `$01 $05 [dummy] [dummy] [data …]`

The "Write Data" command will write to the file that is currently open. If there is no file open, the status channel will read "`85,NO FILE OPEN`". If the file is not opened for writing, the file system will return "`ACCESS DENIED`" onto the status channel. The command will never return data.

The two dummy bytes are there to align the data on a long-word boundary. The suggested transfer size is 512 bytes at a time. This will give the optimal performance, while keeping into consideration the maximum command transfer size.

2.6. DOS_CMD_FILE_SEEK (0x06)

Command format: `$01 $06 [posL] [posML] [posMH] [posH]`

The "File Seek" command places the pointer into the currently opened file at a user-defined position. The command takes one argument: a 32 bit value, which is transferred LSB first.

The command never returns any data. When the seek is successful, status returns "`00,OK`", or else a message from the file system. If there is no file open, the status channel will read "`85,NO FILE OPEN`".

2.7. DOS_CMD_FILE_INFO (0x07)

Command format: `$01 $07`

The "File Info" command returns a data packet with information about the currently open file. In fact, this command executes a file stat command. The format of the data packet is as follows:

```
DWORD   size;  /* File size */
WORD    date;  /* Last modified date */
WORD    time;  /* Last modified time */
char    extension[3];
BYTE    attrib;      /* Attribute */
char    filename[ ];
```

The status response could either be:
"00,OK", "85,NO FILE OPEN", or "88,NO INFORMATION AVAILABLE"

2.8. DOS_CMD_FILE_STAT (0x08)

Command format: `$01 $08 <filename>`

The "File Info" command returns a data packet with information about a file, specified by the 'filename' parameter. The format of the data packet is the same as for DOS_CMD_FILE_INFO (0x07).

The status response could either be:
"00,OK", or "88,FILE NOT FOUND"

2.9. DOS_CMD_CHANGE_DIR (0x11)

Command format: `$01 $11 <directory name>`

The 'Change Directory" command is used to let the DOS enter a sub directory. When the DOS starts, the current directory will be the root of the SdCard. The parameter given is the name of the directory to enter. Like Windows, Linux and MacOS, the names "." and ".." have special meaning: current and parent directory.

With this command, it is also possible to enter files that have sub-entries, such as ".D64" files. These files are treated as sub file systems, and therefore commands as "File Open" will also work on files within.

This command does never return any data. The status channel will tell whether the operation was successful. The two possible responses are: "00,OK", or "83,NO SUCH DIRECTORY".

2.10. DOS_CMD_GET_PATH (0x12)

Command format: `$01 $12`

The "Get Path" command will return the current path in the file system, starting from the root. The path string is returned as a data packet. The status channel reports "00,OK", as this command can never fail.

2.11. DOS_CMD_OPEN_DIR (0x13)

Command format: `$01 $13`

The "Open Directory" command will attempt to start reading the current directory. The command will not return any data, but it will return status information: "`00,OK`", "`01,DIRECTORY EMPTY`", or, if there was an error: "`86,CAN'T READ DIRECTORY`".

2.12. DOS_CMD_READ_DIR (0x14)

Command format: `$01 $14`

The "Read Directory" command will return the contents of the directory to the data channel. Each entry of the directory is transmitted as a data packet. The format is simple: The first byte gives the attribute of the directory entry, followed by the file name. The attribute has the following fields:

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
		ARCHIVE	DIR	VOLUME	SYSTEM	HIDDEN	RE-ADONLY

These fields are taken from the attribute byte as it exists in FAT directories, and is reused for other non-FAT directories.

2.13. DOS_CMD_COPY_UI_PATH (0x15)

Command format: `$01 $15`

The "Copy User Interface Path" is issued to continue using the path that was already opened in the file browser. This enables the use of relative paths in the software.

The command is executed and then falls through to the "Get Path" command; thus it will return the current path which the file browser is at.

2.14. DOS_CMD_LOAD_REU (0x21)

Command format:
`$01 $21 [addrL] [addrML] [addrMH] [addrH] [lenL] [lenML] [lenMH] [lenH]`

The "Load REU" command can be used to read data from the currently opened file into the REU memory. The command takes two 32-bit parameters, both LSB first. The first argument is the REU address at which the data is loaded, the second gives the total number of bytes that shall be read. The load function does not wrap around; the load is truncated when the start address plus the length exceeds the end address of the REU memory. The upper bytes of both the address as well as the length are masked out, thus effectively these bytes are dummy bytes.

Note: This function assumes a 16 MB REU configuration.

The status message is either "`00,OK`", "`02,REQUEST TRUNCATED`", or a message directly from the file system.

The data that is returned is a more detailed string, indicating the number of bytes read at which address, such as: "`$003000 BYTES LOADED TO REU $126800`".

2.15. DOS_CMD_SAVE_REU (0x22)

Command format:
```
$01 $22 [addrL] [addrML] [addrMH] [addrH] [lenL] [lenML] [lenMH] [lenH]
```

The "Save REU" command can be used to write data to the currently opened file from the REU memory. The command takes two 32-bit parameters, both LSB first. The first argument is the REU address from which the data is saved; the second gives the total number of bytes that shall be written. The save function does not wrap around; it is truncated when the start address plus the length exceeds the end address of the REU memory. The upper bytes of both the address as well as the length are masked out, thus effectively these bytes are dummy bytes.

Note: This function assumes a 16 MB REU configuration.

The status message is either "`00,OK`", "`02,REQUEST TRUNCATED`", or a message directly from the file system.

The data that is returned is a more detailed string, indicating the number of bytes written from which address, such as: "`$008000 BYTES SAVED FROM REU $852000`".

2.16. DOS_CMD_ECHO (0xF0)

Command format:
```
$01 $F0
```

This command will simply echo the command back as a data packet. The status channel will return "`00,OK`", as this command cannot fail.