

Cassiopei

USB communication

(firmware release 20140112 or higher)

Table of Contents

| | | |
|------|---|----|
| 1 | Introduction..... | 3 |
| 2 | Why the Cassiopei uses the USB HID-class..... | 4 |
| 3 | Constant definitions..... | 5 |
| 4 | Version and status commands..... | 6 |
| 4.1 | Version of the Cassiopei's firmware..... | 6 |
| 4.2 | Status of the Cassiopei's TAP player..... | 7 |
| 4.3 | Remaining space of flash filesystem..... | 8 |
| 5 | File transfer from PC to Cassiopei..... | 9 |
| 5.1 | Adding a file to the filesystem..... | 9 |
| 5.2 | The transfer of a virtual PRG file..... | 11 |
| 5.3 | The transfer of a virtual TAP file..... | 13 |
| 6 | File managing actions..... | 14 |
| 6.1 | Read file info..... | 14 |
| 6.2 | Delete current file..... | 15 |
| 6.3 | Read current file..... | 16 |
| 6.4 | Count real file size..... | 18 |
| 7 | Raw flash memory access..... | 19 |
| 7.1 | Read raw flash memory data..... | 19 |
| 7.2 | Write raw flash memory data..... | 20 |
| 8 | TAP related commands..... | 21 |
| 8.1 | Start playback of current TAP file..... | 21 |
| 8.2 | Stop playback of current TAP file..... | 21 |
| 8.3 | Change position inside current TAP file..... | 22 |
| 9 | EEPROM commands..... | 23 |
| 9.1 | EEPROM read..... | 23 |
| 9.2 | EEPROM write..... | 23 |
| 10 | Special flash related commands..... | 24 |
| 10.1 | Erase flash memory..... | 24 |
| 11 | Unused and reserved for future use..... | 25 |

1 Introduction

This document describes the basic set of commands required to communicate with the Cassiopei. This information is usefull for anyone who wants to write his/her own implementation of the “Cassiopei manager” software.

2 Why the Cassiopei uses the USB HID-class

One of the things I hate the most is devices that are used for many years and then are removed from their USB port somewhere on the back of your computer, sometime later you plug them back into a different port and suddenly the computer no longer recognizes the device you've used many times before. A problem that is quickly solved by selecting the correct device driver... but that is for the situation where you have this driver at a convenient location (a CD or the internet), but what if you do not have access to the internet and the CD is lost somewhere in the house.

The USB HID-class is a class used by many input devices, joysticks, gamepads, keyboards, your mouse. All those devices that want to transfer small amounts of data. The Windows OS will not request a driver it will not search on the internet, this class is so well known/defined it is already embedded into the OS. So isn't that convenient?

There are many kinds of classes in the USB protocol, another class that would have been practical is the mass-storage class. Using this class the Cassiopei could have been designed like a USB-stick, thumb-drive, hard-disk, etc. That would have been nice, as these devices do not really require a program to work, the OS is the tool to upload files onto the device. Unfortunately this class has the disadvantage that everything has to be like a file making it a bit tedious/difficult to do other things than transporting files. Also the OS might attempt communication with the Cassiopei at moments that the Cassiopei isn't ready for it. This situation would make it difficult for the programmer (that's me) to implement all the functions of the Cassiopei in a practical manner. Also the mass-storage class does require a device driver, normally this would be already available to the OS but there are no real guarantees. Because IF the Cassiopei would be a mass-storage device then it would be connected to Windows, Linux, Apple devices, Android systems most likely even Amigas. And this would mean that I would have to support all these different types of computers and operating systems, while all that I really want to do is make a simple device for use with my Windows computer and C64 (and PET3032, VIC20, C128, C16). So in short, no mass-storage device.

The Cassiopei is a slave device. The Windows PC is the master device. In USB the master talks to the slave, the slave only answers when spoken to. This makes it a bit difficult to do things but practically it is not a real problem. One simple rule is the following, do not transmit data to the master (PC), because of you do when the master (PC) did not request it... it gets lost.

The PC sends a command to the Cassiopei and the Cassiopei responds. For instance, we want to transfer a file to the Cassiopei's flashmemory. Then the following happens. The PC sends the file transfer request command, the Cassiopei receives this and prepares itself for filetransfer, the Cassiopei responds with "ready for transfer", the file is transmitted in packages, the Cassiopei acknowledges each package and by doing so the PC knows when to send the next package and so on.

3 Constant definitions

Because we are a HID device we can transmit 64 bytes per packet (the absolute max for a HID packet). But this does not mean that we use all 64 bytes for data transfer. Some packets require additional overhead (for example a memory address of 3 bytes)), below are the definitions used by the cassiopei for the size of the different types of packages

PACKET_PAYLOADSIZE_DATAFILE = 58

PACKET_PAYLOADSIZE_USERFILE = 63

PACKET_PAYLOADSIZE_VIRTUALFILE = 61

Attention: changed from 56 to 61 since firmware release 20140112

BYTESPERBLOCK = 4096 - 4

A block has 4096 available bytes of which 4 a used by the filesystem (3 bytes for pointing to the next block, plus 1 byte to indicate that it is the first or a following block)

BYTESPERFILEHEADER = 44

A fileheader consists of ... bytes that describe the details of the file (name, size, timestamp, etc)

4 Version and status commands

4.1 Version of the Cassiopei's firmware

The version information is very usefull for the Cassiopei manager to check if the connected Cassiopei is capable of executing the given commands. For example, new firmware and Cassiopei manager software is distributed, the user installs the Cassiopei manager and uses that, then the Cassiopei manager would do very unexpected things if the Casiopei has not been updated with the proper firmware to supportthe new functionality. So therefore the Cassiopei manager checks the firmware version of the connected Casiopei and if the detected version may causes compatibillity problems... then the Cassiopei manager shows a message stating a firmware update is required and disabled the Cassiopei manager. A simple but effective method to prevent problems due to users not reading the manuals or instructions.

The package send by the PC looks like this:

```
byte 0      : 0x02 (value indicates version request)
byte 1-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x02 (value indicates version information package)
byte 1      : YEAR value (0-99), add 2000 to get the real year value
byte 2      : MONTH value (1-12)
byte 3      : DAY value (1-31)
byte 4-63   : unused
```

4.2 Status of the Cassiopei's TAP player

The status information is very usefull for the Cassiopei manager to check visualize the current status of the Cassiopei. This function is used by the TAP file player functions. To shows the progress of the “tape”. So the counter is updated and the user gets feedback about the status of the progress being made. Simply because nothing is more frustrating then waiting without knowing how long it takes AND if knowing if any progress is being made.

The position counter is a 3 byte counter, allowing to navigate through a TAP file of 16MBytes. Considering the Cassiopei has only 8MByte of flash, this is more then sufficient. The value indicates the byte position (not the pulses), this because counting bytes is much easier then counting pulses, simply because bytes are allways bytes and pulses are 1 byte or 4 bytes in size. So the position counter counts bytes. When the pointer has the value 0, it points to the first data byte of the TAP file (which is the first byte after the header in the flash filesystem on the Cassiopei). So when the pointer reaches the value identical to the filesize, the file has reached the end.

The package send by the PC looks like this:

```
byte 0      : 0x03 (value indicates status request)
byte 1-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x03 (value indicates status of TAP player package)
byte 1      : undefined
byte 2      : high byte of 3 byte tape counter
byte 3      : .. byte of 3 byte tape counter
byte 4      : low byte of 3 byte tape counter
byte 5      : tape motor status (0=motor OFF, 1=motor ON)
byte 6      : current mode-value of the Cassiopei
byte 7      : current index-value of the Cassiopei
byte 8-63   : unused
```

4.3 Remaining space of flash filesystem

The Cassiopei uses a file system to keep track of the free blocks on the flash memory. However it might be the case that this filesystem does not match with the information read from all the fileheader of all the sotred files on the filesystem. This is easily achieved by writing a file to the Cassiopei and then during that procedure... disconnecting or aborting the write action. In that case the fileheader states a file of (in example) 24 blocks, while in reality only 12 blocks are written before the user interrupted the write action. If the Cassiopei manager want to know what the real free space is on the Cassiopei's flash memory, the counting all the sizes from the individual file headers isn't enough. So the flash needs to be scanned block by block to determine the number of really free blocks. This seems like a time consuming task, but in reality you hardly notice.

The package send by the PC looks like this:

```
byte 0      : 0x05 (value indicates remaining space request)
byte 1-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x05 (value indicates remaining space package)
byte 1      : low byte of the 2 byte remaining blocks counter
byte 2      : high byte of the 2 byte remaining blocks counter
byte 3-63   : unused
```


5 File transfer from PC to Cassiopei

5.1 Adding a file to the filesystem

The Cassiopei has an 8MByte of flash memory and this needs to be filled with all sorts of files. The following procedure demonstrates how a .PRG file must be transferred to the Cassiopei.

First we must tell the Cassiopei we want to transfer a file. We do this by sending the file details in the first package. This package send by the PC looks like this:

```
byte 0      : 0x20 (prepare the Cassiopei for receiving a file)
byte 1      : filetype (0-255)
byte 2      : filename (ASCII) byte 1
byte ..     : filename (ASCII) byte ..
byte 33     : filename (ASCII) byte 32
byte 34     : filesize MSB
byte 35     : filesize .SB
byte 36     : filesize LSB
byte 37     : Timestamp YY..      (example 20 for the year 2014)
byte 38     : Timestamp ..YY      (example 14 for the year 2014)
byte 39     : Timestamp Month     (1-12)
byte 40     : Timestamp Day       (1-31)
byte 41     : Timestamp Hour      (0-23)
byte 42     : Timestamp Minutes   (0-59)
byte 43     : 0x00                 (reserved for future use)
byte 44     : 0x00                 (reserved for future use)
byte 45     : 0x00                 (reserved for future use)
byte 46     : Low byte of last addr. of program in CBM memory
byte 47     : High byte of last addr. of program in CBM memory
byte 48     : Low byte of LOAD addr. of program in CBM memory
byte 49     : High byte of LOAD addr. of program in CBM memory
byte ?-63   : unused
```

The bytes 34-36 describe the filesize in blocks (not in bytes). It is used to be able to detect file(s) that are partially written (because of a problem during writing, for instance the Cassiopei is disconnected from the USB port). The calculation of this value is to be calculated using the formula below:

```
'calculate the number of packages that need to be transmitted
If (filesize Mod payloadsize = 0) Then
    numberofpackages = filesize \ payloadsize
Else
    numberofpackages = (filesize \ payloadsize) + 1
End If
extra_padding_bytes = (numberofpackages * PACKET_PAYLOADSIZE_USERFILE) - filesize

If (((filesize + BytesPerFileheader + 1 + extra_padding_bytes) Mod BytesPerBlock) = 0) Then
    Calculate_FileSizeSizeInBlocks = ((filesize+BYTESPERFILEHEADER+extra_padding_bytes)\BYTESPERBLOCK) * BYTESPERBLOCK
Else
    Calculate_FileSizeSizeInBlocks = (((filesize+BYTESPERFILEHEADER+extra_padding_bytes)\BYTESPERBLOCK)+1) * BYTESPERBLOCK
End If
```

The bytes 46-49 that specify the last address and the LOAD address are specifically for .PRG files. Other filetype do not require this information. Please refer to the description of the filetypes in the Cassiopei user manual to learn more about how these files are written to flash.

The Cassiopei creates a new file with the given data stored in the file header and responds with a package that looks like this:

```
byte 0      : 0x20 (Cassiopei acknowledges it is ready for transfer)
byte 1-63   : unused
```

The first package holds the filesize and startaddress and the remaining part of the package is filled with the data of the file. The maximum size of an HID package could be 64 bytes, the Cassiopei uses `PACKET_PAYLOADSIZE_USERFILE + 1` bytes (+1 because of the command code 0x21). The first virtual file package looks like this:

```
byte 0      : 0x21 (value indicates file package)
byte 1      : low byte of last addr of program in CBM RAM
byte 2      : high byte of last addr of program in CBM RAM
byte 3      : low byte of LOAD addr of program in CBM RAM
byte 4      : high byte of LOAD addr of program in CBM RAM
byte 5      : data byte 0
byte 6      : data byte 1
byte 7      : data byte 2
byte ..     : ...
byte PACKET_PAYLOADSIZE_USERFILE : last byte in this package
```

The Cassiopei will respond with an acknowledge. Which is nothing more then the following package:

```
byte 0      : 0x21 (value indicates file package)
byte 1-63   : unused
```

This is the signal for the PC application to transmit another package of data that looks like :

```
byte 0      : 0x21 (value indicates file package transfer succeeded)
byte 1      : data byte
byte 2      : data byte
byte 3      : data byte
byte ..     : ...
byte PACKET_PAYLOADSIZE_USERFILE : last byte in this package
```

The Cassiopei will respond with an acknowledge. Which is nothing more then the following package:

```
byte 0      : 0x21 (value indicates file package)
byte 1-63   : unused
```

And this keeps on repeating until the entire file has been transmitted. The last package might have not enough data to fill the package. In that case the package needs to be padded with 0x00's. These 0x00's will also be written to the Cassiopei's flash memory.

Note:

`PACKET_PAYLOADSIZE_USERFILE` is a constant, see the chapter “constants” for it's value.

5.2 The transfer of a virtual PRG file

The Cassiopei must be set into virtual TAP file mode. This can be done by using the following method:

- Press shift+runstop on the CBM computer (or type LOAD <enter> for machines that do not support shift+runstop).
- The computer states “press play on tape”
- Press the “Menu” button on the Cassiopei
- The menu will now load, select the virtual PRG file mode and exit the menu
- The virtual PRG file mode is now being entered, the Cassiopei will load the fastloader routine and as soon as that is ready the actual file data is to be transferred.

The first package holds the filesize and startaddress and the remaining part of the package is filled with the data of the file to be transmitted. Although the maximum size of an HID package could be 64 bytes the Cassiopei uses this package to it's full extend. The number of “payload” bytes is limited to 63 bytes. The first virtual file package looks like this:

```
byte 0      : 0x22 (value indicates virtual PRG file package)
byte 1      : low byte of last addr of program in CBM RAM
byte 2      : high byte of last addr of program in CBM RAM
byte 3      : low byte of LOAD addr of program in CBM RAM
byte 4      : high byte of LOAD addr of program in CBM RAM
byte 5      : data byte 0
byte 6      : data byte 1
byte 7      : data byte 2
byte 8      : ...
byte 9      : ...
byte PACKET_PAYLOADSIZE_VIRTUALFILE : last byte in package
```

The Cassiopei will receive the first package from the PC and when it has processed the data it will respond with a acknowledge package. Which is nothing more then the following package:

```
byte 0      : 0x22 (virtual PRG file package processed)
byte 1-63   : unused
```

This is the signal for the PC application to transmit another package of data that looks like :

```
byte 0      : 0x22 (value indicates virtual PRG file package)
byte 1      : data byte
byte 2      : data byte
byte 3      : data byte
byte 4      : ...
byte 5      : ...
byte PACKET_PAYLOADSIZE_VIRTUALFILE : last byte in package
```

Again wait for the acknowledge and then send another package and keep repeating this process of sending and waiting for acknowledge until the entire file has been processed (or the Cassiopei

disconnects). When the last package is to be created, the situation will occur that not all “payload” locations of the package are used. In that case the package must be filled with 0x00's.

5.3 The transfer of a virtual TAP file

The Cassiopei must be set into virtual TAP file mode. This can be done by using the following method:

- Press shift+runstop on the CBM computer (or type LOAD <enter> for machines that do not support shift+runstop).
- The computer states “press play on tape”
- Press the “Menu” button on the Cassiopei
- The menu will now load, select the virtual TAP file mode and exit the menu
- The virtual TAP file mode is now being entered, the Cassiopei will now wait for the virtual TAP to be transferred.

Basically a TAP file is nothing more then a series of tones, described very accurately. This makes it easy to transmit. We need to specify to the Cassiopei what kind of TAP file we are sending, so to keep things easy we send this information in every packet. A Virtual TAP packet looks like this.

```
byte 0      : 0x23 (value indicates virtual TAP file package)
byte 1      : TAP file version (supported values are: 0,1)
byte 2      : data
byte 3      : data
byte ..     : ...
byte PACKET_PAYLOADSIZE_VIRTUALFILE : last byte in package
```

The Cassiopei will receive the first package from the PC and when it has processed the data it will respond with a acknowledge package. Which is nothing more then the following package:

```
byte 0      : 0x23 (virtual TAP file package processed)
byte 1-63   : unused
```

This is the signal for the PC application to transmit another package and this process repeats until the user stops the transfer or the file has reached the end. Because this is a virtual file, the positioning inside the TAP file can all be done on the PC side. Simply by sending the required data through the following packages. There is no need to request status information regarding TAP playback, because this info is already available at the PC side, since this is sending the data.

6 File managing actions

The functions in this chapter are mainly for managing existing files on the filesystem.

6.1 Read file info

To build a directory of the files present on the filesystem. Or in other words, to make a list of the files and their details as found on the flash memory, the command read-file-info is available. We can read the filename, size, filetype, etc. This command is used for searching through the filesystem to build up a directory, but also to search for the file that needs to be deleted.

The package send by the PC looks like this:

```
byte 0      : 0x51 (value indicates a file info request)
byte 1      : fileindex (0-253)
byte 2      : filetype (0-255)
byte 3-63   : unused
```

The filetypes and how these files are written to flash are described in the Cassiopei user manual. Therefore all possible filetypes are not to be described in this manual.

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x51 (value indicates raw flash read data packet)
byte 1      : 0=file not found, 1=file found
byte 2      : fileheader data byte 0
byte 3      : fileheader data byte 1
byte 4      : ...
byte 5      : ...
byte 49     : fileheader data byte 47
byte 50     : high byte of flashaddres of first block of this file
byte 51     : ... byte of flashaddres of first block of this file
byte 52-63 : unused
```

The flash address of the first block of this file is actually a 3 byte value. But since the low byte is always 0x00, there is no real reason for sending it. The flash address of the first block of the file is very usefull for debugging purposes. If there are doubt about the contents of the file, simply use this value and enter in into the flash monitor of the Cassiopei manager, then you can examine the raw flash data.

6.2 Delete current file

The proper procedure for deleting a file uses this command. This command deletes the current file. So we first must make sure that the current file is selected. This can be done by requesting the file info (see command in the previous chapter). Then we can compare the file details with the file we want to delete and if these match we issue the delete command as shown below.

The package sent by the PC looks like this:

```
byte 0      : 0x52 (value indicates a delete current file request)
byte 1-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x52 (value indicates delete has succeeded)
byte 1-63   : unused
```

6.3 Read current file

Based on the information we have from reading the file info, we know the size of the file. Therefore we can read it and know when the file ends. This function allows the reading of the file data. We can read the current file, so just like the delete function we must search for the file we want to read. Once we have found the file we want, we issue the read-current-file command.

The package send by the PC looks like this:

```
byte 0      : 0x53 (value indicates a read current file request)
byte 1-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x53 (value indicates read result package)
byte 1      : data from file
byte 2      : data from file
byte ..     :
byte PACKET_PAYLOADSIZE_USERFILE : last byte
```

Now we have read `PACKET_PAYLOADSIZE_USERFILE` bytes from the file, we need to send another read-current-file command the get the next package and another one and another one until all bytes form the file are read.

Note:

PACKET_PAYLOADSIZE_USERFILE is a constant, see the chapter “constants” for it's value.

6.4 Count real file size

In order to verify the size of a file (checking of the file is as large as specified in the file header) a counting command is required. This command is useful to determine which file has “corrupted” the file system. For instance, when a file is written but during the writing stage, the Cassiopei is disconnected. Then the file is written only partially. This means that the header specifies more blocks as used than there are actually used. So there is an inconsistency of the filesize and therefore the remaining space of the flash memory. By comparing the filesize as defined in the fileheader with the filesize as given by this routine, we can determine which file is incomplete and needs to be removed (or repaired).

The package sent by the PC looks like this:

```
byte 0      : 0x54 (value indicates a count file size request)
byte 1      : fileindex (0-253)
byte 2      : filetype (0-255)
byte 3-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x54 (value indicates file count result)
byte 1      : low byte of the number of blocks used by this file
byte 2      : high byte of the number of blocks used by this file
byte 3-63   : unused
```

7 Raw flash memory access

The functions in this chapter are mainly for system files and diagnostics of the flash memory. These routines are not intended to be used for adding .PRG, TAP, etc. files to the filesystem. This because these routines completely bypass the filesystem.

7.1 Read raw flash memory data

During development of the Cassiopei's filesystem, there was a need for reading back the raw data from the flash memory. And with raw data I mean every byte from every block. So this is the file data including the filesystem data. Or in short all data!

The package send by the PC looks like this:

```
byte 0      : 0x41 (value indicates a raw flash read request)
byte 1      : high byte of the 3 byte address value
byte 2      : ... byte of the 3 byte address value
byte 3      : low byte of the 3 byte address value
byte 4-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x41 (value indicates raw flash read data packet)
byte 1      : data byte 0
byte 2      : data byte 1
byte 3      : data byte 2
byte 4      : ...
byte 5      : ...
byte 32     : data byte 31
byte 33-63 : unused
```

7.2 Write raw flash memory data

In some cases absolute files are required (.SYS a.k.a. system files), in example, the speech synthesis routines use speech samples and in order to improve the performance of the playback routines, absolute files are required. This is simply a system that does not use any filesystem overhead. Without a filesystem we can be certain that data is written to absolute flash address. Or in other words we do not have to be afraid of fragmented files as the files are written as a continuous stream of data. Therefore we can calculate the absolute address of each byte in that file. And therefore make it very easy addressable. This way we do not need to rely on file searching routines or need to compensate for long searching delays, which would result in clicks, ticks and pauses in the generated speech. The only drawback to this method is that the filesystem needs to be completely erased before we can write data using this method. Once a system file has been written, other files using the filesystem may be added.

However, in combination with the read function as described in the previous chapter, it would be possible to completely backup the flash memory contents AND write them back if required.

The package sent by the PC looks like this:

```
byte 0      : 0x42 (value indicates a raw flash write request)
byte 1      : high byte of the 3 byte address value
byte 2      : ... byte of the 3 byte address value
byte 3      : low byte of the 3 byte address value
byte 4      : data byte 0
byte 5      : data byte 1
byte 6      : data byte 2
byte 7      : ...
byte 8      : ...
byte 35     : data byte 31
byte 36-63 : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x42 (value indicates raw flash write succeeded)
byte 1-63   : unused
```

8 TAP related commands

8.1 Start playback of current TAP file

To allow the Cassiopei manager to controll the TAP functionality of the Cassiopei this function was implemented. Sending this commands starts the playback of the currently selected TAP file.

The package send by the PC looks like this:

```
byte 0      : 0x72 (value indicates play request)
byte 1-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x72 (value indicates play request was received)
byte 1-63   : unused
```

8.2 Stop playback of current TAP file

To allow the Cassiopei manager to controll the TAP functionality of the Cassiopei this function was implemented. Sending this commands stops the playback of the currently selected TAP file.

The package send by the PC looks like this:

```
byte 0      : 0x73 (value indicates stop request)
byte 1-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x73 (value indicates stop request was received)
byte 1-63   : unused
```

8.3 Change position inside current TAP file

To allow the Cassiopei manager to controll the TAP functionality of the Cassiopei this function was implemented. Sending this commands allows to change the current position inside the TAP file. The position counter is a 3 byte counter, allowing to navigate through a TAP file of 16MBytes. Considering the Cassiopei has only 8MByte of flash, this is more then sufficient. The value indicates the byte position (not the pulses), this because counting bytes is much easier then counting pulses, simply because bytes are allways bytes and pulses are 1 byte or 4 bytes in size. So the position counter counts bytes. When the pointer has the value 0, it points to the first data byte of the TAP file (which is the first byte after the header in the flash filesystem on the Cassiopei). So when the pointer reaches the value identical to the filesize, the file has reached the end.

The package send by the PC looks like this:

```
byte 0      : 0x74 (value indicates position request)
byte 1      : high byte of the 3 byte position value
byte 2      : ... byte of the 3 byte position value
byte 3      : low byte of the 3 byte position value
byte 4-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x74 (value indicates position request was received)
byte 1-63   : unused
```

9 EEPROM commands

The Cassiopei has no internal EEPROM and therefore the term EEPROM isn't technically correct, however, the useage of these fields resemble the functionality of EEPROM memory. By using a special method of wear levelling (see the Cassiopei user manual) the flash memory is made to behave like it has EEPROM registers. For the easy of reading/programming, this functionality is referred to as (simulated) EEPROM. To keep it simple, the max number of EEPROM locations is limited to 15 bytes. Which is more then enough to store the Cassiopei related settings. **This functionality is not intended to be used by user programs !!** Simply to prevent compatibillity problems when future functionality is added to the Cassiopei.

9.1 EEPROM read

To read the current setting as stored in flash memory. The package send by the PC looks like this:

```
byte 0      : 0x61 (value indicates simulated EEPROM read request)
byte 1      : location (0-14)
byte 2-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x61 (value indicates simulated EEPROM read)
byte 1      : the value read from simulated EEPROM
byte 1-63   : unused
```

9.2 EEPROM write

To write a setting to flash memory. The package send by the PC looks like this:

```
byte 0      : 0x62 (value indicates simulated EEPROM write request)
byte 1      : location (0-14)
byte 2      : the value to be written to the simulated EEPROM
byte 3-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0x61 (value indicates sim. EEPROM write succeeded)
byte 1-63   : unused
```

10 Special flash related commands

10.1 Erase flash memory

In some cases, it is easier to wipe the entire flash memory and start from scratch building up a new set of files. For instance when you decide to use the Cassiopei for a different computer model and do not want to remove 230 games one at a time. And offcourse for development of the Cassiopei, it was a very usefull function to clear the entire flash.

The package send by the PC looks like this:

```
byte 0      : 0xF0 (value indicates flash erase request)
byte 1-63   : unused
```

The Cassiopei responds with a package that looks like this:

```
byte 0      : 0xF0 (value indicates flash erase succeeded)
byte 1-63   : unused
```


11 Unused and reserved for future use

The two commands below are currently not used, and reserved for future use

CMD_PREPARE_FILE_TO_PC 0x30 /*prepare transfer from Cassiopei to PC*/

CMD_FILE_TO_PC 0x31 /*a file packet from this device to the PC*