

Cassiopei

USER MANUAL

this manual is intended for
Cassiopei firmware version: V20160213 or higher
Cassiopei manager (PC software) version: V20160213 or higher

the expansion for all Commodore computers with a cassetteport

Table of Contents

1	Introduction.....	4
2	User manual.....	5
2.1	The Cassiopei.....	6
2.2	Upgrading the Cassiopei's firmware.....	9
2.3	Configuration.....	11
2.3.1	System requirements.....	11
2.3.2	Visibility of the Cassiopei on Windows XP.....	12
2.3.3	Visibility of the Cassiopei on Windows 7.....	13
2.3.4	Visibility of the Cassiopei on Windows 8.1.....	14
2.3.5	Using the Cassiopei manager.....	15
2.3.6	Using the Cassiopei menu program.....	23
2.3.7	Using the Cassiopei SetCompModel.....	24
2.4	Supported computer models.....	26
2.4.1	PAL/NTSC or 50/60Hz system compatibility.....	26
2.4.2	Commodore 64.....	27
2.4.3	Commodore VIC-20.....	28
	What to do when a VIC-20 program does not run?.....	30
2.4.4	Commodore C16/Plus4.....	34
2.4.5	Commodore 128.....	35
2.4.6	Commodore PET/CBM 20XX series (build-in tape drive).....	37
2.4.7	Commodore PET/CBM 20XX and 30XX series.....	39
2.4.8	Commodore PET/CBM 40XX series.....	41
2.4.9	Commodore PET/CBM 80XX series.....	43
2.4.10	Commodore CBM 600 series (not supported).....	45
2.5	General usage of the Cassiopei.....	46
2.5.1	Compatibility.....	46
2.5.2	LOAD and SAVE.....	48
2.5.3	TAP file usage.....	49
2.5.4	Cross development.....	51
2.5.5	BASIC wedge (C64).....	52
2.5.6	BASIC wedge (C128).....	53
2.5.7	BASIC wedge PET/CBM3000 series.....	54
2.5.8	BASIC wedge for all other models.....	56
2.5.9	Speech synthesizer.....	57
3	Technical details.....	58
3.1	Tape protocol of the CBM's kernal loader.....	59
3.1.1	Signal encoding.....	59
3.1.2	Polarity of the signal.....	60
3.1.3	How a byte is recorded.....	61
3.1.4	Structure on the tape.....	62
3.1.5	Data on the tape.....	65
3.1.6	Digital flywheel technology.....	66
3.2	How the fastloader is loaded.....	67
3.3	CPIO protocol.....	69
3.3.1	The CPIO protocol's signals (synchronous 8-bit).....	70
3.3.2	The CPIO protocol's signals (asynchronous 4-bit).....	72
3.4	CPIO messages definitions.....	73
3.4.1	Command: Load.....	74
3.4.2	Command: Data load.....	75

3.4.3	Command: Data save.....	76
3.4.4	Command: File info from flash.....	78
3.4.5	Command: Get remaining space.....	78
3.4.6	Command: Simulate button.....	79
3.4.7	Command: Read ADC.....	80
3.4.8	Command: Read / Write EEPROM.....	81
3.4.9	Command: DTMF generator.....	82
3.4.10	Command: Speech synthesizer.....	83
3.4.11	Command: Sample player.....	85
3.4.12	Command: Sinewave generator.....	88
3.4.13	Command: KlikAanKlikUit (first generation).....	89
3.4.14	Command: KlikAanKlikUit (second generation).....	91
3.4.15	Command: I2C.....	94
3.4.16	Command: ServoController commands.....	98
3.5	Cassiopei filesystem.....	99
3.5.1	The flash memory's file system.....	99
3.5.2	System files and user files.....	102
3.5.3	Speech sample file.....	103
3.5.4	EEPROM file.....	105
3.5.5	Vocabulary file.....	108
3.5.6	Program file.....	109
3.5.7	TAP file.....	110
	TAP file compression.....	111
3.5.8	WAV file.....	114
3.5.9	DAT file.....	115
3.6	Expansion connector.....	116
3.6.1	Required Cassiopei expansion connector.....	117
3.6.2	Voltage levels of the Cassiopei.....	118
3.6.3	Reset input.....	119
3.6.4	Analog inputs (ADC).....	120
3.6.5	Audio output (PWM).....	121
3.6.6	I ² C bus.....	122
	Reserved I ² C addresses.....	122
	Addressing of I ² C devices on the I ² C bus using the Cassiopei's BASIC wedge:.....	122
	I ² C devices you could use for your project.....	123
	RC hobby servo controller using the PCA9685.....	124
3.6.7	KlikAanKlikUit signals.....	126

1 Introduction

Hello, my name is Jan Derogee. I am the proud owner of a PAL Commodore 64-II and other Commodore computers. The C64 was my very first computer and I have fond memories of many many days of happy computing with it. Although the C2N Datasette was sufficient at first (it was cheap), I was very happy once I've got my first disk drive. Years later I've bought an Amiga 500 then a 1200 and finally a PC and then another one, and another one, etc. All computers fulfilled my needs but the C64 will always be the model which is the closest to my heart.

When I bought an SX-64 (thanks to a tip of a co-worker), I tumbled into the retro scene. I designed the 1541-III which was the first “fully functional .D64 MMC/SD-card reader emulating a real drive using the IEC-bus, in a case”. I learned a lot from that project and it got my retro fire burning. I'm no gamer I'm just a technician who loves to build the things that were impossible or “out of reach” expensive during the glory days of the C64. Now with today's micro-controller technology things are much easier.

After the success of the 1541-III, it became time for something new, A device that would work on ALL 8-bit commodore computers. Simply because many 8-bit Commodore computers do not have an IEC-bus. The Cassiopei is a device that can connect to all 8-bit Commodore computers. By using the cassetteport as it's physical interface to the computer and by using dedicated software for controlling this port, the Cassiopei becomes a versatile tool for every kind of user. The Cassiopei is perfectly suited for the programmer, the tinkerer, the gamer and the collector.

I would like to thank:

My family, for allowing me to work on my project during the evenings and weekends.

The retro community: HCC Commodore gebruikers groep

(<http://www.commodore.hcc.nl>), this community brings Commodore enthusiasts together. The club meetings (6 times a year) are always motivating and inspire me with new project ideas and are a great testing ground for trying out new things.

Arthur Jordison for making “CBM program studio” the perfect tool for ASM and BASIC coders.

Luigi Di Fraia for http://c64tapes.org/dokuwiki/doku.php?id=loaders:rom_loader

Gideon Zweijtzter for info about “Cass Read is inverse of Cass Write” and RUN related info

Saleae: (<http://www.saleae.com>) for the perfect low cost high performance logic analyzer.

Rigol: for building the perfect low cost oscilloscope (DS1052E).

And all the people who worked at/for Commodore and created these wonderful machines and their documentation. In order to honor these people this document prefers the spelling of the word kernel as “kernal”. A simple mistake that that was copied and later adopted by other authors in many following documents. For more info about this word, search the internet for “kernal”.

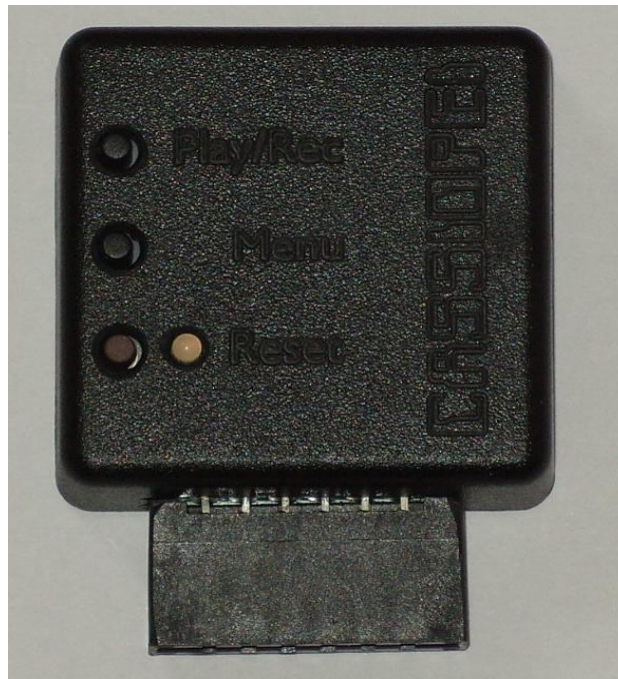
Cassiopei

user manual

2.1 The Cassiopei

Cassiopei stands for: Cassette IO Peripheral Expansion Interface. You can load single file games and programs and connect your own hardware projects. When properly configured it is the only device you'll need on your 8-bit computer.

Here you see a picture of how the Cassiopei looks like, there is the edge connector at the side of the Cassiopei, which allows it's to connect to your CBM's cassette port. Because almost all 8-bit CBM computers feature such a port (C16 and plus4 require the standard CBM conversion connector) the Cassiopei becomes a very useful device. The basic idea about the CPIO expander is that it is (once properly configured) completely self supporting. It does not need any storage device (tape, floppy, harddisk, flashcard, CD, etc.) in order to operate. You can use it to play games, you can use it to load programs. You can use it to connect modern electronics of your own design with great ease of use. The Cassiopei is also great for cross development due to it's quick and easy transfer of data to your CBM computer.



Buttons:

- Play/Rec:** When the computer says “press play on tape” you press this button (it loads the file as selected previously in the menu program).
When the computer says “press record and play on tape” you press this button.
- Menu:** When the computer says “press play on tape”, but you want to choose the file you want to load, then you press this button. The computer will now load the menu file.
- Reset** The computer itself cannot reset the Cassiopei, therefore this needs to be done manually. The Cassiopei resets also when the CBM's power is cycled (but this requires the Cassiopei to be disconnected from USB, otherwise USB will continue to power the Cassiopei and the reset will not take place).

LED:

Located directly next to the reset button is a status LED. The table below shows the different colors and it's causes.

LED	
Color	Information
-	No USB connection, no file activity
Blinking red/green	Bootloader mode (press reset to exit this mode)
Green	Loading file (using kernal tape protocol)
Blinking green	Loading file(using kernal tape protocol) has paused (press CBM key) or stopped because loading has finished
Red	USB activity (IDLE state of USB packet polling)
	USB packets are transferred
	CPIO activity (communication between CBM computer and Cassiopei)
Orange	After reset the Cassiopei briefly lights the LED to indicate reset situation
	After pressing play for TAP file playback, the LED lights to indicate that the Cassiopei is searching for the requested position in the TAP file.
Orange, red, green, off, orange, red, green, off, orange, red, green, off, etc.	<p>The selected file could not be found on the filesystem. This situation occurs when the wrong index has been selected or when the menu file for the selected computer model isn't installed onto the Cassiopei filesystem.</p> <p>This situation is a PANIC situation, the Cassiopei will “lock up” and must be resetted using the reset button. It can however still be accessed via the Cassiopei manager.</p>

USB connector:

This connector requires a mini USB connector (A male to mini USB B male). This connector is supplied along with your Cassiopei, it's length is 3 feet but longer cables are allowed. If the Cassiopei is also connected to a CBM computer at the same time you may use a USB extension cord of up to 5 meters. However this is not recommended as it's reliability cannot be guaranteed due to the wide variation in available extension cables.

The Cassiopei uses USB to connect to a PC. This connection is required to configure the Cassiopei and to load the desired programs/games into the on-board flash memory of the Cassiopei. Once the Cassiopei is configured/filled with files it does not need to be connected to a PC anymore. It can perfectly function as a standalone device. During the time it is connected to a PC it does not need to be connected to a CBM computer. Therefore the PC and the CBM computer do not need to be close to each other allowing more freedom in its usage.



The CBM computer and the PC can be connected to the Cassiopei if desired, this is very useful in cross development situation, where software is developed on the PC and needs to be tested on the CBM computer. For this situation the “virtual file mode” is implemented, more about this mode further on in this manual.

Expansion connector:

This connector makes it possible to connect the Cassiopei to your own hardware projects. The Cassiopei features an I2C port, with this port it is connect to a broad range of integrated circuits. The technical details of the expansion connector are described in the last chapter of this manual.

This little 10-pin port is indicated by the letters PEI in the name Cassiopei. As it defines the Peripheral Expansion Interface.

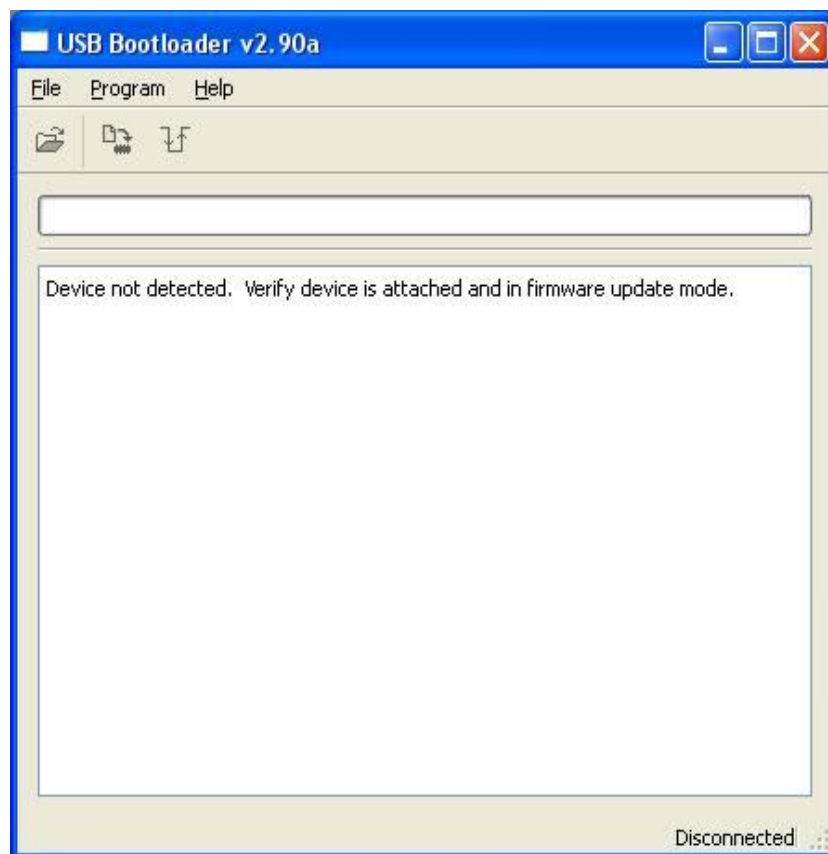


2.2 Upgrading the Cassiopei's firmware

The Cassiopei holds a bootloader program that makes it possible to upload new firmware versions. If the bootloader mode is entered by accident, do not worry, just press the reset button and the Cassiopei will function normally.

You can find a link to the latest version of the Cassiopei's firmware on the Cassiopei project page <http://home.kpn.nl/bderogee1980/projects/Cassiopei>

In order to upload new firmware into the Cassiopei, a program on the PC is also required as well as the .HEX file with the new firmware release. This program is called “USB bootloader v2.90a” and can also be downloaded from the Cassiopei website. When the program is started it may look like the image on the left, all buttons are grayed out because there is no connection with the Cassiopei.



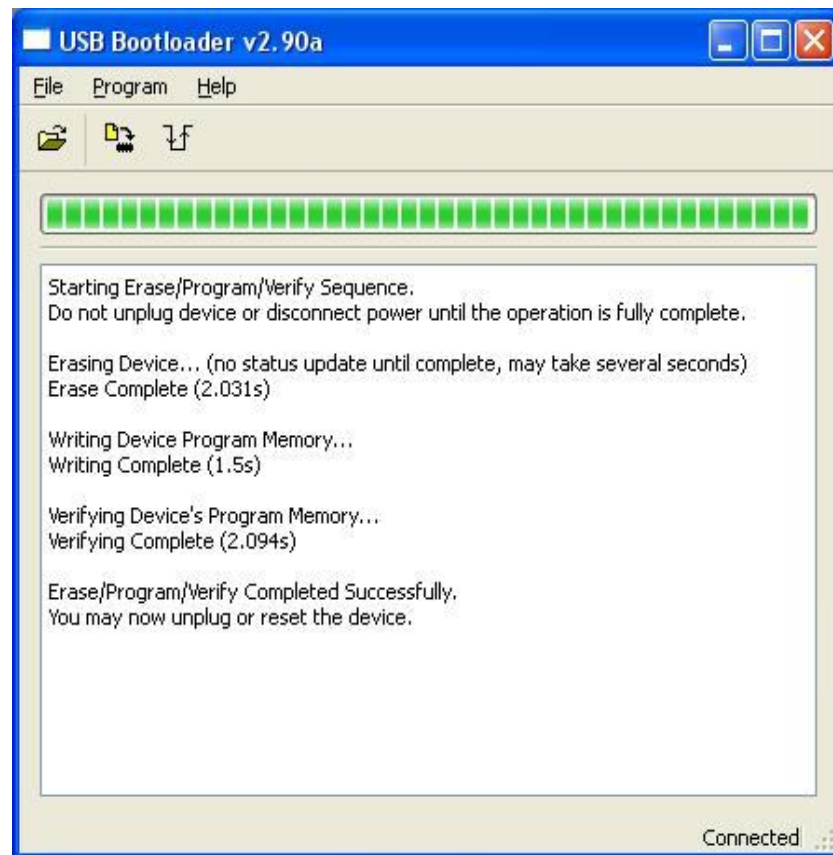
The Cassiopei must be connected to the PC using the supplied USB cable. Then it must be set into the bootloader mode. You can do this by the following steps:

- Hold down both the menu AND reset buttons of the Cassiopei
- Keep holding down the menu button while releasing the reset button
- 1 Second later you release the menu button
- The LED on the Cassiopei will flash RED/GREEN to indicate that the bootloader is active

The PC application will become enabled and you may now program the firmware:

- Select the .HEX file: File → Import firmware image → choose file
- Select: Program → Erase/program/verify device

Now the programming will take place, this takes only a few seconds, the progress bar indicates the action. When programming is ready you may reset the device, using the Cassiopei reset button. Then you may disconnect the USB cable. The Cassiopei has been updated and is ready for use.



Note:

Programming new firmware does not affect the contents of the Flash memory that stores your games and programs, however it is always a good idea to have a backup of important games and programs.

During the programming of the firmware it is important that the Cassiopei is not reset or disconnected. If this does happen, the Cassiopei's firmware would be incomplete and programming has to be done again. The bootloader inside the Cassiopei cannot be destroyed therefore "bricking" your Cassiopei is impossible (unless you use a hammer or lightning bolt).

Do not attempt to program new firmware into the Cassiopei using alternative software!

2.3 Configuration

2.3.1 System requirements

The system requirements in order to run the Cassiopei Manager and related software are:

- Windows XP, SP3 (32 bit), Windows 7 (32 or 64 bit), Windows 8 (32 or 64 bit)
- Enough free space (1GByte) on your harddisk to allow the application and the .net framework 4 client profile to be installed
- Internet access as the PC requires to download certain .net libraries from the Microsoft website. When the installation has finished, internet access is no longer required.
- If using Windows XP then you require direct access to a root USB port (see the “Attention” below about Article ID: 940021) (simply do not use a USB-HUB but one of the USB ports at the back of the computer)

Attention (Windows XP users):

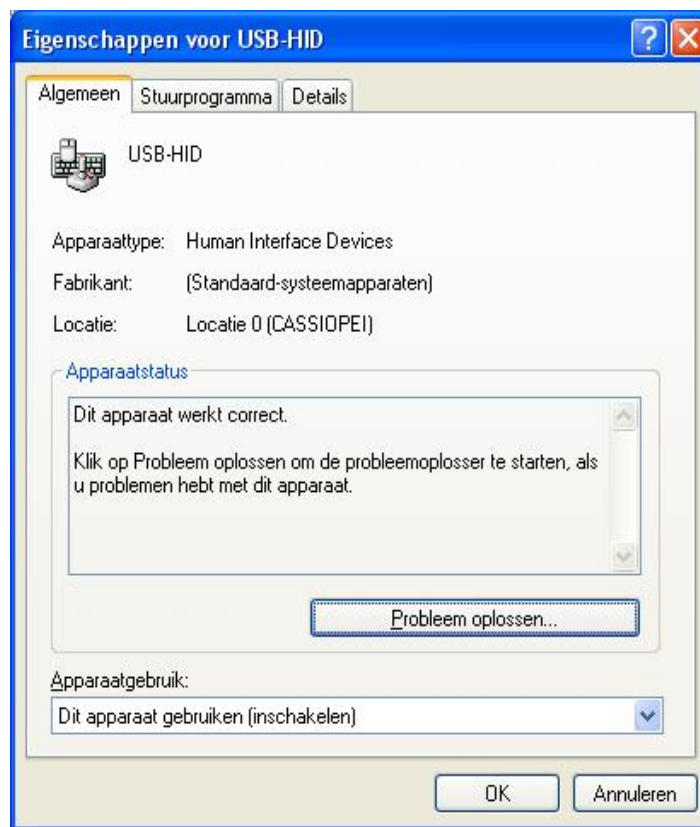
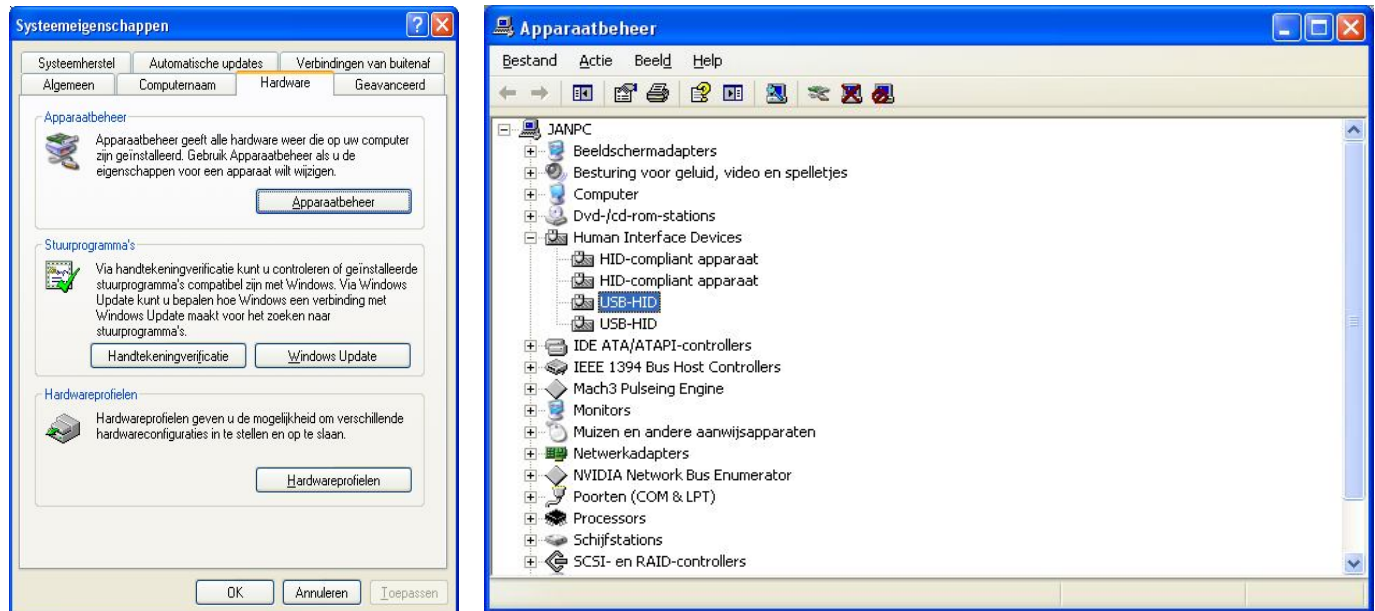
It is possible to transfer file with an effective speed of 21000 byte/sec. But you may experience a much slower transfer speed. In that case you might suffer from a “bug” in the Windows XP HID driver (Article ID: 940021), which results in bad performance when a 1.x USB device is connected to a 2.0 hub. This can be solved by connecting the Cassiopei directly to one of the USB ports at the back of your PC. The USB ports at the front of your computer could be connected to an internal USB hub, so be aware.

Although a slower transfer speed might not be a real problem for uploading files to the Cassiopei's flash memory, it is a huge problem for the Virtual TAP file mode. This because TAP files have a very critical timing and when the data from the buffers are used up faster then they are added then a buffer underflow occurs and the TAP file transfer is corrupted. The effective buffer time in virtual TAP file mode is only 10 mSec (for a signal of 2500Hz). Enlarging the buffers is no solution.

2.3.2 Visibility of the Cassiopei on Windows XP

When the Cassiopei is connected to the PC on a Windows XP system the Cassiopei will be visible as a normal HID device in the Hardware properties on your machine. Below are some screen shots of a Dutch version of Windows XP showing the Cassiopei. This menu shows when you right-click on the my computer icon on your desktop and choose the properties.

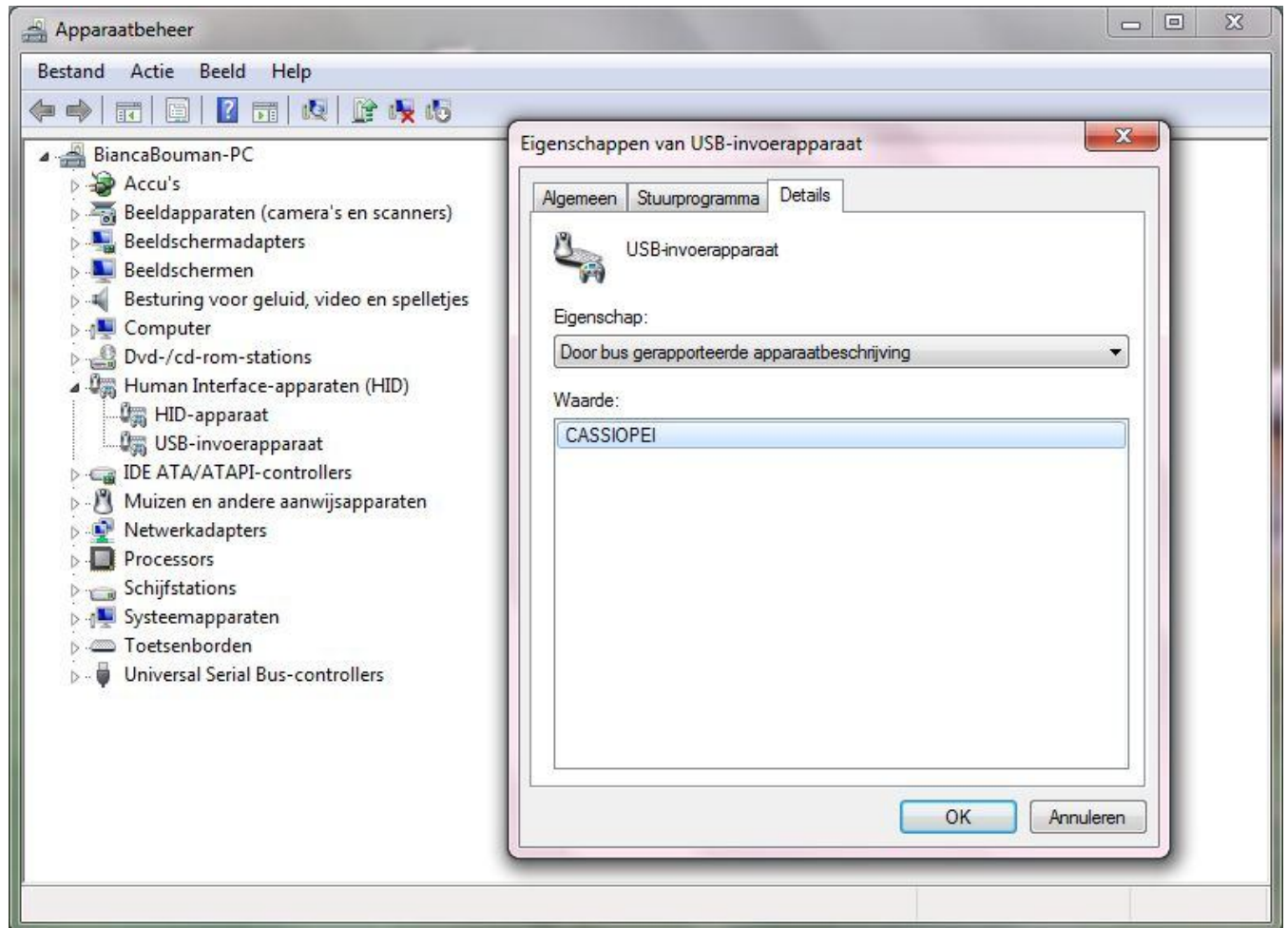
You can also go to Start → Settings → configurations → System



2.3.3 Visibility of the Cassiopei on Windows 7

When the Cassiopei is connected to the PC on a Windows 7 system (service pack 1) then Cassiopei will be visible as a normal HID device in the Hardware properties on your machine. Below are some screen shots of a dutch version of Windows 7 showing the Cassiopei in the “Apparaatbeheer” which is Dutch for “device manager”. This menu shows when you right-click on the my computer icon on your desktop and choose the properties. You can also go to “Start” → Device manager

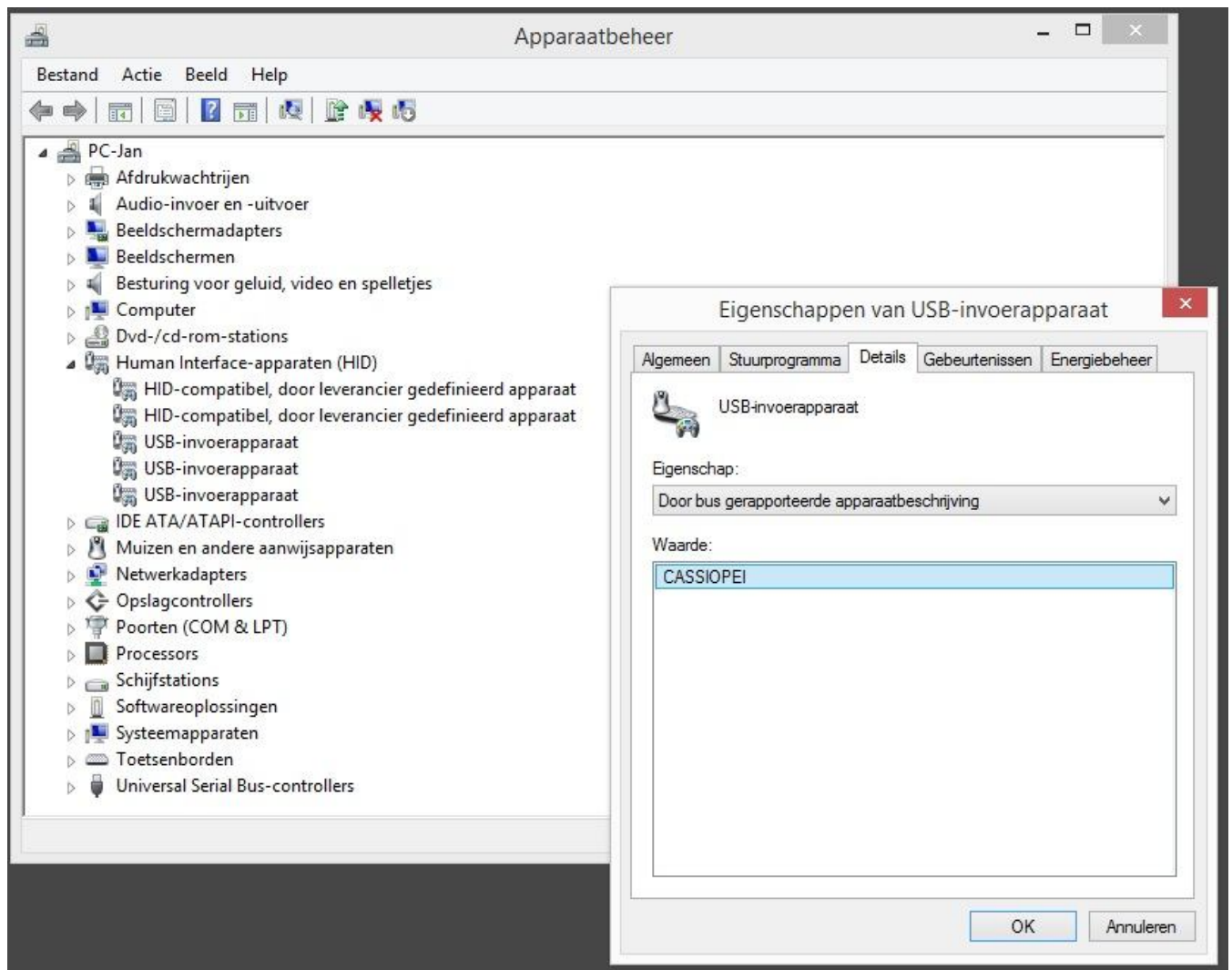
In the details tab it is possible to show the name of the device selected. Unfortunately, where in windows XP this was directly visible, you now need to select the proper value that holds the name. In order to see the real name of the device you have to change the “eigenschap” (“property”) to “door bus gerapporteerde apparaatbeschrijving” (which means something like “reported device description”).



2.3.4 Visibility of the Cassiopei on Windows 8.1

When the Cassiopei is connected to the PC on a Windows 8.1 system then Cassiopei will be visible as a normal HID device in the Hardware properties on your machine. Below are some screen shots of a dutch version of Windows 8.1 showing the Cassiopei in the “Apparaatbeheer” which is Dutch for “device manager”. This menu shows when you right-click on the my computer icon on your desktop and choose the properties. You can also go to “Start” → Device manager

In the details tab it is possible to show the name of the device selected. Unfortunately, where in windows XP this was directly visible, you now need to select the proper value that holds the name. In order to see the real name of the device you have to change the “eigenschap” (“property”) to “door bus gerapporteerde apparaatbeschrijving” (which means something like “reported device description”).

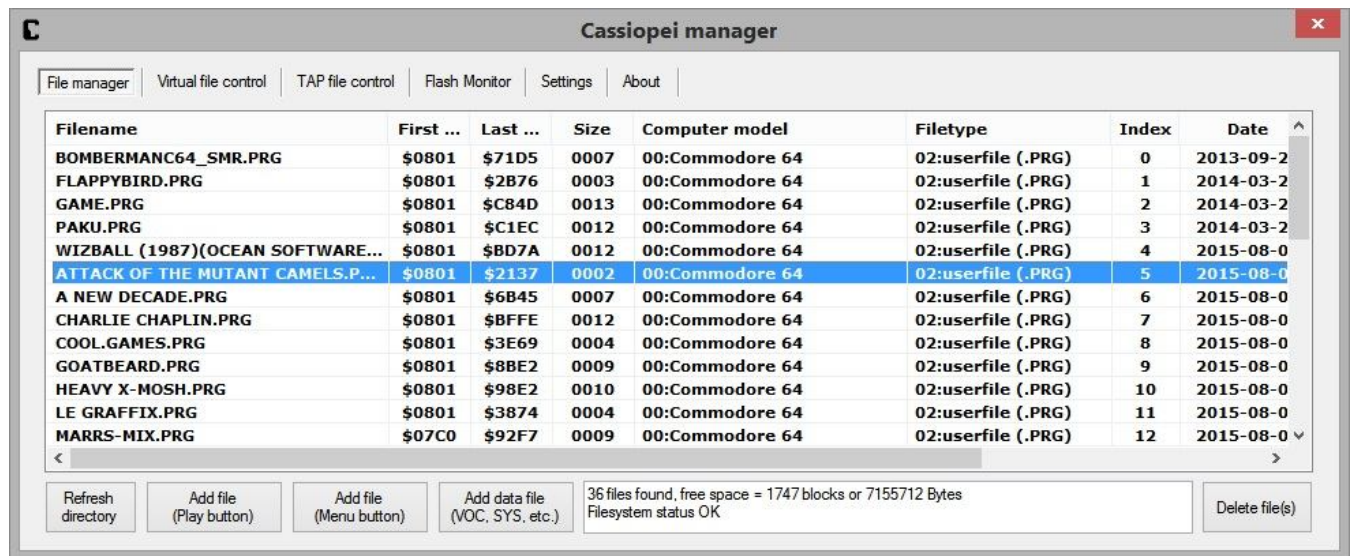


2.3.5 Using the Cassiopei manager

In order to configure the Cassiopei, the Cassiopei manager program must be installed onto your PC. This program transfers the data from the PC to the internal Flash memory of the Cassiopei and it is used to select the computer the Cassiopei is used with. This program consists of 4 screens which will be explained on the next few pages. The Cassiopei manager is not supplied with the Cassiopei, but must be downloaded from the Cassiopei website.

Before starting the Cassiopei manager make sure that the Cassiopei is connected to the PC using the supplied USB cable. When the Cassiopei is connected to the USB port of your PC it requires no additional drivers, as all required drivers are build into the Cassiopei manager.

Files:



The screen-shot shows the file screen, this screen is used to transfer files into the Cassiopei's internal Flash memory. You can add a file by clicking on one of the "add file" buttons. There are 3 buttons the Add file play, menu and data file button. The play button is used for your games/programs and the menu button is used for the Cassiopei menu program. There can be many games/programs added to the system. But there can be added only one menu file. Or to be more precise, one menu-file per computer model. In order for the Cassiopei to work, you must install a Cassiopei menu program for the computermodel(s) you intend to use it on. Because of the differences of the various Commodore computers each computermodel requires it's own unique Cassiopei menu program. The Add data file button is intended to upload data files to your Cassiopei, these files can be system files (holding speech data), VOC speech vocabulary files, WAV audio files or disk image files (for example D64 files that must be transferred to a real disk).

The Cassiopei can be connected to various computermodels. And if you like to use the Cassiopei for more then one computer, then it isn't very practical to connect it to a PC and reconfigure. Therefore you can load files for different computermodels onto your Cassiopei. When you add a file, the Cassiopei will ask you for which computer model. This however can be disabled by changing a setting in the settings tab. If you disable the computermodel question then the Cassiopei manager will assume that every file added is intended for the currently selected computermodel (the computer model that the Cassiopei is configured for).

In the same settings tab is also a setting that makes it possible to hide all the files that are not for the currently selected computermodel. This may improve your overview. The menu program on your Commodore computer that allows you to browse through the files on the Cassiopei, will only show the files for the currently configured computermodel. Therefore if you have installed files for C64, C16 and the VIC-20 and you are working on a VIC-20, then you won't see the files intended for the C64 and C16. This because these files are made invisible for the Cassiopei. This prevents you from confusion when you try to start a game named PACMAN if you have installed it for multiple computermodels.

When you have a totally empty Cassiopei (or have selected a computermodel for which there are no files installed yet), you'll see the Settings file and nothing else. This settings file holds the settings, like mode, file index, etc. It is a system file, you have no control over it.

The Cassiopei's file index counters are 8-bits wide, limiting the max amount of files per filetype per computermodel to 254. In other words, the max. number of files that can be addressed are 254 of each filetype per computermodel. So theoretically, there can be 254 .PRG files and 254 .TAP files stored (for each computermodel). At first this seems like a small number but when you realize that the flash memory of the Cassiopei is "only" 8 Megabytes, the method of addressing will not cause any problems.

Because the Cassiopei can also save files from the CBM computer, it makes sense that there is an "extract .PRG file button", press this button to extract a file from the Cassiopei and save it to your PC's harddisk.

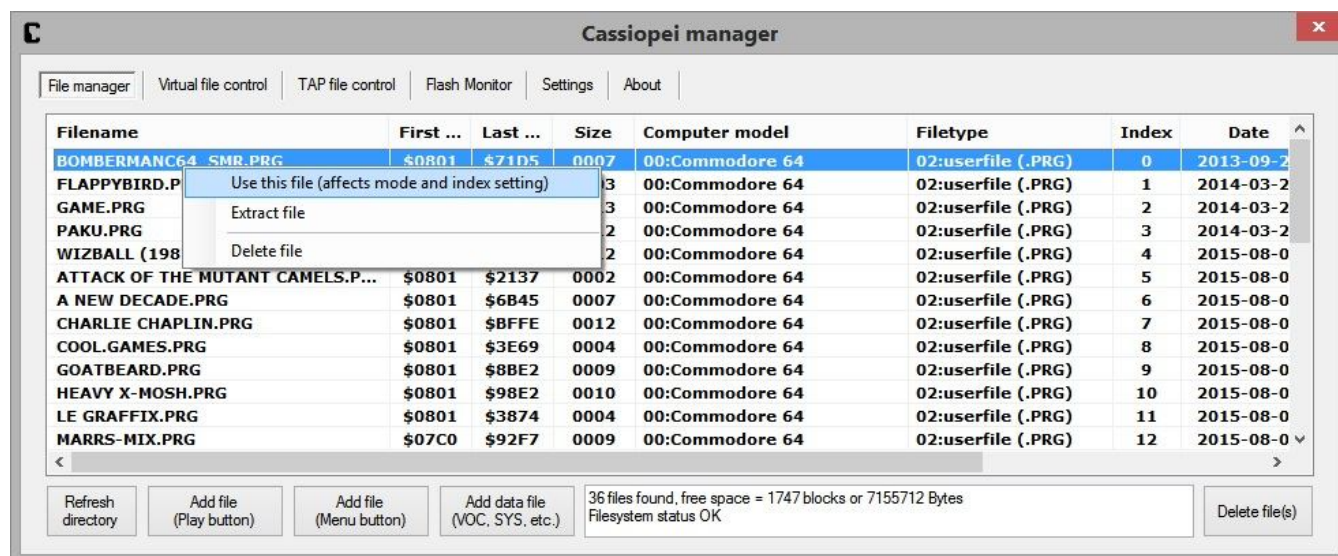
You may also want to remove a file, this is possible by selecting that file and then click "delete file", the file will be deleted from the Cassiopei's Flash memory

The "read directory" button refreshes the file overview window, normally this button is not required as the screen is refreshed after every file action. However if for some reason the USB cable becomes disconnected, pressing this button might help to restore the connection when all cables are reconnected properly again. However it is strongly advised not to disconnect the Cassiopei during filetransfer of any kind.

Attention:

Do not attempt to transfer files from the PC to the Cassiopei while the CBM computer is accessing/using the Cassiopei, as this may lead to file corruption. Also do NOT reset or disconnect the Cassiopei from the PC during file transfer, as this may lead to file corruption. This sounds more scarier than it is. In "the good old days", you didn't remove the disk during writing either, or did you?

Selecting a file is possible by changing the “mode” and “index” in the Settings screen. Although very effective, this method is far from user friendly. To make things easier, the user can also right-click with the mouse on the file. Then a window pops up, see the screenshot below.

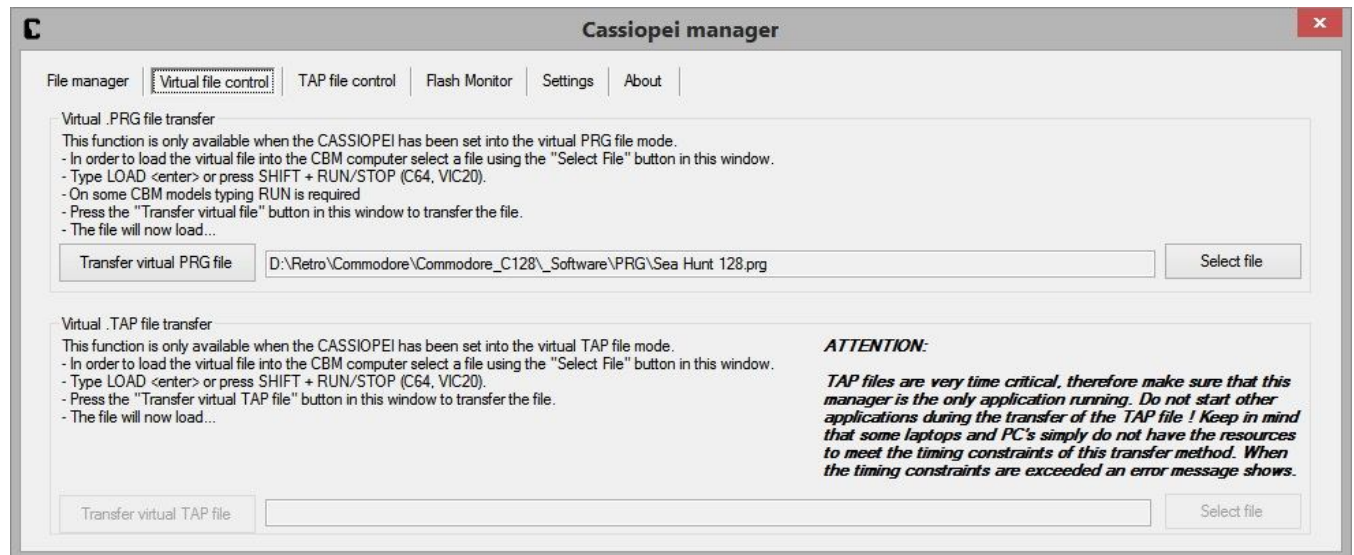


By choosing the option “use this file” the “mode” and “index” value are set to the proper values automatically. So when this is done, the user can type LOAD on the CBM computer and press “play” on the Cassiopei. The file loads now using the Cassiopei's fastloader from flash.

When this function is used the user will be asked if the fastloader should be used. It is suggested that the user answers this question with “yes”. Simply because the file is loaded more then 50x faster then the regular Commodore tape loading routines. Of course loading times are highly depending on the size of the file, so this isn't always a problem. A 1KByte file for any computer doesn't take that long if the fastloader isn't used (so 50x faster doesn't mean that much). But a 50KByte file takes many minutes and isn't really preferred by most users in these modern times.

Also available under the “right-click” menu are “extract file” to save a .PRG file to the PC's harddisk. If the “extract file” option is disabled, then this means that this functionality is not available for the selected file. The “delete file” function removes the selected file from the Cassiopei's flash memory.

Virtual file:



The previous screen showed the files as stored in the Cassiopei's Flash memory, however during development of new games and programs it is desired to quickly upload a file to the CBM computer. Saving a file to the Flash memory and clearing it again because it was wrong would be a waste of time. The “virtual file mode” allows to transfer a file directly into the memory of the CBM computer, saving precious development time and reduces the need to click all those tiny buttons.

The “select file” button is required to select the “virtual file”, the file that will be transferred to the CBM's memory. This functionality only works for .PRG files and uses the Cassiopei's fast loader routines. The “virtual file mode” must be selected using the Settings screen of the Cassiopei manager or by the menu program (the file linked to the menu button of the Cassiopei) on your CBM computer. This mode works as follows, type:

LOAD <return>

The computer will now ask you to “press play on tape”:

- press the PLAY/REC button on the Cassiopei.
- press the “transfer file” button (in the virtual file screen)
- on a C64 press space or CBM key when the filename is shown (or simply wait 10 seconds)
- when the CBM is ready loading the fastloader routines then the transfer starts and the program automatically executes (assuming it can be started using the RUN command, programs that require a SYS command must be started manually).

Usage of command line options:

For development purposes, commandline options would be very practical. These would allow you to combine the functionality of the cassiopei manager inside your development tool(s), this save the user time to click on buttons of the Cassiopei managers GUI..

- configure the Cassiopei for virtual file mode usage (only required if not already configured for this mode, use the cassiopei manager or the menu-program)

-type load (or shift run/stop) on the CBM computer

-....Cassiopei_manager.exe /v d:\test.prg

TAP file control:

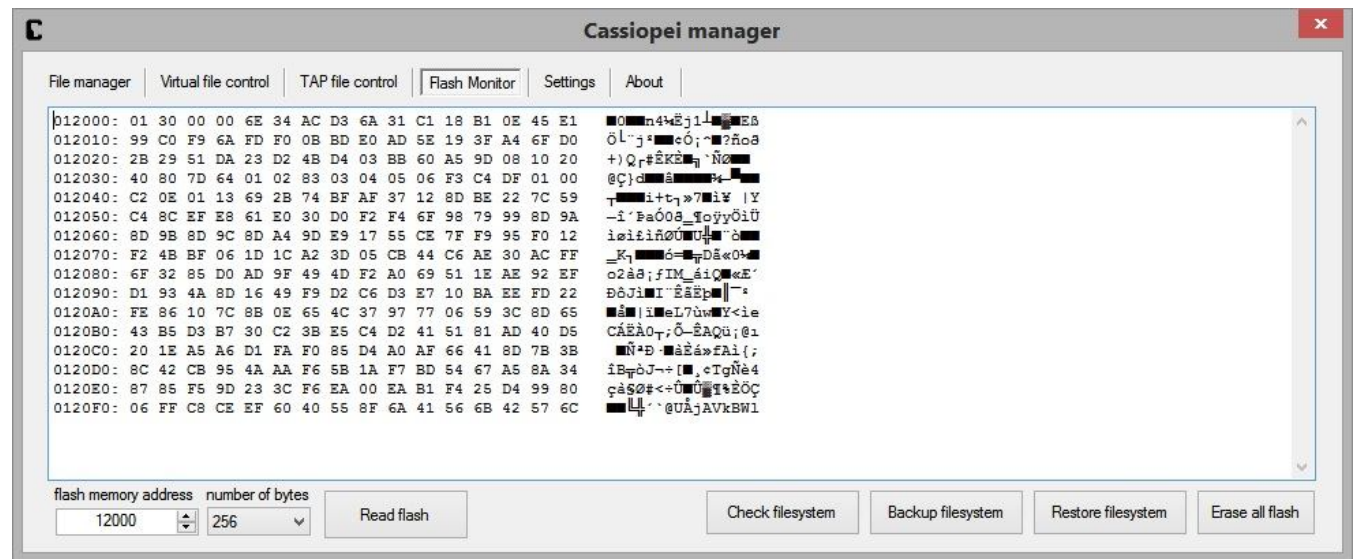
This screen is very useful for playing TAP files that require the user to navigate through the contents of the tape. Because the Cassiopei has only 3 buttons, navigating through the file using these buttons would be far from practical. Therefore the Cassiopei manager has a simulated datasette so you can experience the original tape feeling using the interface you are already familiar with.

This interface is available when the Cassiopei is configured for use with TAP files and when the Cassiopei is configured to use virtual TAP files. Please keep in mind that virtual TAP files are very critical regarding their timing and may not work properly on all computers. If virtual TAP files do not function, then load the TAP file into the Cassiopei and play the TAP from the Cassiopei's flash memory.



Flash monitor:

This screen is intended for development of the Cassiopei but may also come in handy for support issues. Therefore it is fully available to the user, but most users will never have to access this screen.



If for any reason the filesystem has become corrupted, then the corrupted files must be removed in order to restore filesystem integrity, if this is not possible, then press the “erase flash button”. But before you panic... there is a filesystem check function. This allows you to identify the files that are corrupt and it is just a matter of deleting these files. But first let me tell you about the possible problems that you most likely will never experience but theoretically can occur. There are 2 possibilities:

Corrupt files: these are files that are incomplete and therefore are marked larger than they really are. In other words, these files are missing blocks and the end of the file and are therefore considered to be incomplete/corrupt by the filesystem.

Orphaned blocks: these are blocks that are marked by the filesystem as used but do not belong to any file.

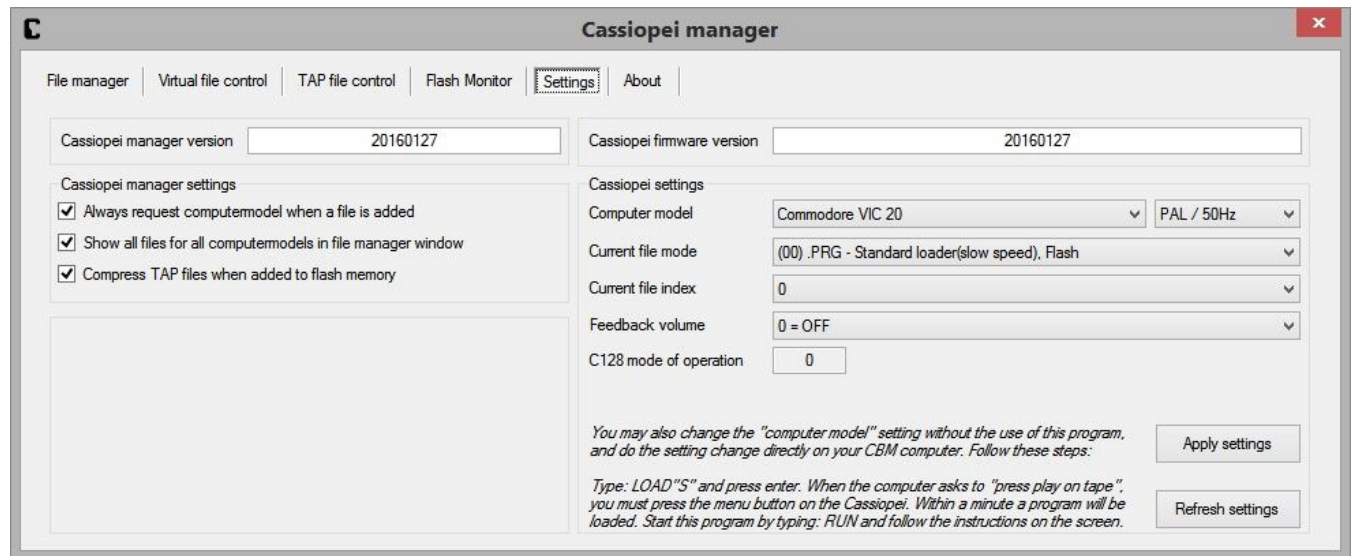
The Corrupt files are mainly caused by users who abort a write action to the flash memory. For instance, the user is uploading a file to the Cassiopei and then disconnects or resets the Cassiopei. The file that is being written is therefore aborted and the end of the file is not on the filesystem. This is comparable to removing a disk from your diskdrive when the computer is still writing to the disk. You simply should not do that. The only way to fix this type of filesystem problems is by deleting these non complete files. When using the Check filesystem function, these files will be marked by a red color in the file manager window.

Orphaned blocks can occur when the user aborts a delete action. Then the beginning of the file is removed and therefore the file itself appears to be removed, but the end of the file may still be on the filesystem. The blocks that remain are still marked as "used" and can no longer be used by the filesystem. Therefore the real free size of the filesystem is not the same as the expected free size and the filesystem. This causes no harm but is a waste of storage space and therefore needs to be repaired. In order to do this the filesystem needs to be scanned. The filechain of every file must be followed to identify the really used blocks. When all files are scanned then the blocks that are not found in any of the files need can be marked as "free". The check filesystem function does this automatically, the only thing the user needs to do is to confirm the deletion of the orphaned blocks.

The flash monitor screen also has 2 buttons that allow you to make a backup copy of your flash filesystem. This action will take approx. 10 minutes and copies the entire Cassiopei filesystem to a file on your PC. Or to restore the filesystem from a backup file. This functionality was originally created as a development tool but because it may come in handy for some users who like to swap between multiple filesystems for whatever reason.

Settings :

This screen is used to configure the Cassiopei for the correct computer model. But it also shows version information about the Cassiopei manager programs and the Cassiopei's firmware.



Computer model:

This setting is the most important one, use it to select your computer model, The computer model is important as there are various subtle differences between the computer models that are supported, by letting the Cassiopei know to what computer it will be connected, the proper timings and IO settings can be taken care of.

The mode setting:

This setting can also be done with the Cassiopei's menu program on the CBM computer.

There are 4 modes of operation:

- **Standard CBM loader (slow speed), flash.** This mode loads PRG files from the internal Flash memory using the standard CBM kernal loader, this is just as slow as a real cassette player.
- **CPIO fastloader (high speed), flash.** This mode loads PRG files from the internal Flash memory using the Cassiopei's fast loader routines (this is the preferred mode).
- **CPIO fastloader (high speed), USB.** This is the virtual file mode, it loads PRG files from the PC's harddisk using the Cassiopei's fast loader routines (this mode is very useful for cross development).
- **Play .TAP file (slow speed), flash.** This mode, plays back images of real cassettes from the internal Flash memory.

Index setting:

This setting holds the index of the current file. The Cassiopei uses an index to keep track of the current file. It does not use filenames when using the menu program on the CBM computer the index is converted back into a filename, so you do not notice. However if files are deleted then the index of the

file that is the current file might change and you end up loading the wrong file. But no data is damaged, so simply use the menu program or this tool to select the proper index and it is solved. The idea behind indexes is speed, searching through 8MByte of flash memory using indexes goes a whole lot faster than when searching for a filename. And most users never notice the difference.

Feedback volume:

It is possible to make the tape signals audible, so you can hear them when you LOAD or SAVE using the standard tape protocol of your CBM computer. This can be useful for debugging or the fun of listening to those nostalgic noises. But most importantly, hearing that your computer is loading or saving gives you an idea of the progress of the file transfer. This sound is switched off by default.

This functionality will direct the tape signals of LOAD and SAVE to the PWM output. Meaning that you need to connect a speaker to the PWM output as described in the chapter regarding the Expansion connector. However, if you decide to connect this output to the input of your C64's SID chip then there is something you should know. The C64 has the volume of the SID set to the lowest level after reset/power-on. So (unless a program has changed the SID volume level) you will hear nothing. You can set the volume of the SID to the max. using the following command: POKE 54296,15

2.3.6 Using the Cassiopei menu program

If the Cassiopei has been configured properly using the Cassiopei manager, then you are able to use the Cassiopei menu program (linked to the MENU button of the Cassiopei). This program will be used frequently to select the game or program that you want to run.

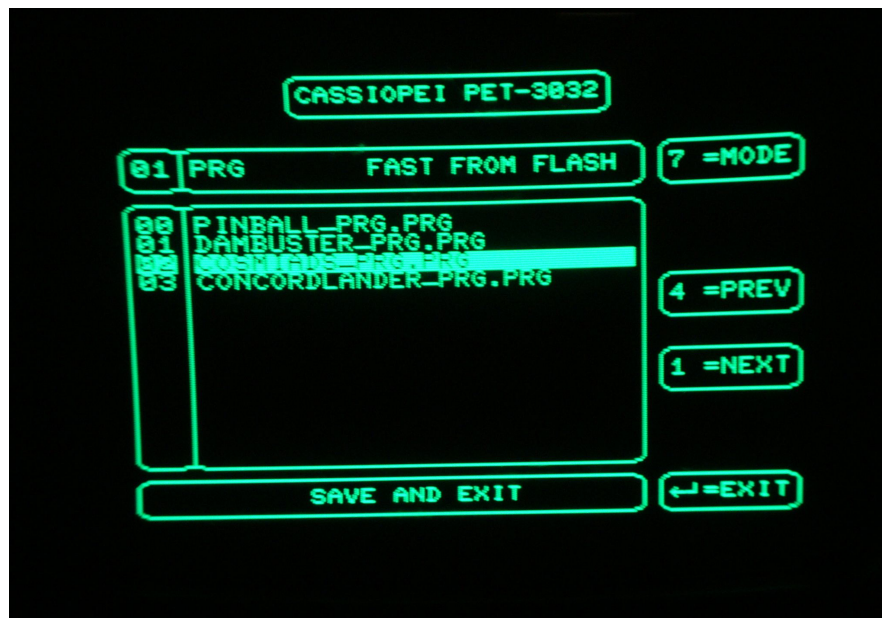
In the situation where you want to start a game or program that you've runs previously, then the menu program is not required, simply press the Play/Rec button when the computer states “press play on tape”. In all other cases follow the instructions below.

Starting (loading) the Cassiopei menu program:

- Type LOAD
- When stated “press play on tape”, press the MENU button on the Cassiopei
- The menu program will appear within a few seconds

Below the screen-shot of a PET 3032 Cassiopei menu program. The layout of this screen is similar for all supported computer types, there are minor details (due to screen size/color limitations) but their functionality is identical.

Use the keys as indicated in the screen to browse through the different operating modes and files. As shown in the screen-shot below, the file with index 02 (COSMIADS_PRG.PRG) will be loaded from flash memory using the Cassiopei's fastloader. By pressing the RETURN key these settings are saved and the menu program exits and automatically loads the chosen program.



2.3.7 Using the Cassiopei SetCompModel

Sometimes you do not have the ability to use the Cassiopei manager PC program in order to configure the Cassiopei for a different computer. Or perhaps you simply don't want to. For example, your PC is in another room or there is no need to upload new games to the Cassiopei, you only want to change the computer model. For these situations there is the Cassiopei SetCompModel program. This is a simple program that is stored in the firmware of your Cassiopei (so you cannot remove it and it will not show up in the Cassiopei manager's file manager window). Loading this program is very easy. Loading this program takes approx. 25 seconds. However loading this program on a C16/plus4 takes slightly longer (a loading time of approx. 50 seconds) because the program is first send using the original timings (which the C16/plus4 can't use) of the Commodore tape protocol and then again but with the timings for the C16/plus4 models.

LOAD"SETCOMPMODEL" <return>
(you may shorten this to: **LOAD"S"** <return>)

The computer will now ask you to "press play on tape":

- press the MENU button in order to load the program

The computer will load the file from the Cassiopei, but because (at this stage) the Cassiopei emulating a real tape it cannot know which file it needs to load. So the Cassiopei sends the configured menu program first (which will be ignored by the computer because the filenames do not match) then it will send the Cassiopei SetCompModel program. This is a BASIC program capable of running on all CBM models. When it has finished loading you must type RUN and the program will start. It will look like the PET/CBM3032 screenshot below. On other computer models only the color will be different.



You will be asked for the computer model and the video system, when confirmed the settings will be send to the Cassiopei (in the form of a file, so it is a bit slow and takes a "few" seconds). When this is done the Cassiopei is configured to the selected computer model.

If you want to verify the new settings, then you can connect the Cassiopei to a PC and start the Cassiopei manager. Or if already started and connected to a PC, press the "refresh directory" in the file manager window. Go to the settings windows you'll see that the settings have changed to the new settings.

Note:

Make sure that the Cassiopei already has the correct files uploaded. For example, the Menu program and the games you want to play. You can configure the Cassiopei to hold all menu programs for all available computer models (or only for the models you intend to use), the Cassiopei will use the correct program depending on your computer model selection.

For every model of computer you ever intend to use you require to install the menu program on the Cassiopei, so when you press the menu button the Cassiopei can load it. Although the C16 and the Plus4 can use the same menu program, it must be installed twice, once for the C16 and once for the Plus4, this because the Cassiopei sees these almost identical computers as 2 different machines each requiring their own menu program. But in practice you can get around this, simply configure the Cassiopei for a Plus4, if you are only switching between a C16 and Plus4. Because if you do so, then you never need to change the computer model settings.

2.4 Supported computer models

2.4.1 PAL/NTSC or 50/60Hz system compatibility

The Cassiopei is a device that can be connected to many computer models and from all the models that Commodore made there is a PAL or NTSC version. And for the Version with a build in monitor, there is a distinction between 50 or 60HZ.

The Cassiopei manager allows you to select PAL/NTSC or 50/60Hz when you select the computer model in the settings tab. This settings is important as it determines the timing for the .TAP file playback and for the loading of the fastloader program. However, once the Cassiopei is in fastloader mode almost all data transfers are synchronous and it no longer matters what system you use.

A feature of the Cassiopei that is coupled to the video system is the sample playback. This will mean that the sample playback rate is related to the frequency of the video system. For example on a C64, or C128. This because the timing is directly related to the line frequency in order to avoid bad lines. If problems arise on NTSC systems, then it may help to disable the screen using poke 53265,11 (C64/C128) before playing the sample and enable the screen using poke 53265,27 (C64/C128) when the sample has finished. This is not an issue when the sample is only played back using the Cassiopei's PWM output at the expansion connector.

2.4.2 Commodore 64

The C64 is the computer that inspired the design of the Cassiopei. As you can see it is easily connected to the cassetteport of both C64 models. It cannot be connected to the C64 games system (as this is a cartridge based game console version of the C64) and it cannot be connected to the SX-64 or DX64, which also has no cassetteport.



C64 "old model"



C64 "new model"

How to use:

Configure the Cassiopei to be used on a C64 computer, this is required in order to define all timings of the Cassiopei for this type of computer, see the chapter "Configuring the Cassiopei". Do not attempt to use the Cassiopei while it is not configured.

The Cassiopei's requires the following sequence of commands independent of it's operating mode.

LOAD <return>

or

shift + RUN/STOP

The computer will now ask you to "press play on tape":

- press the PLAY/REC button on the Cassiopei to load the currently selected game/program
- press the MENU button in order to load the menu program to define your new settings.

The Cassiopei menu program is to be used for selecting the file (when usage of the Cassiopei manager is not possible or desired). Navigating through the menu is possible using the keyboard or by moving the joystick (port 1 or 2) up/down and fire to select.

When a program is loaded using the Cassiopei's fastloader, the program is executed directly after it has finished loading. However .TAP files do no use the Cassiopei's fastloader and depending on the .TAP file used it may be required to type in RUN.

2.4.3 Commodore VIC-20

The VIC-20 is the predecessor of the C64 and has many limitations. However the Cassiopei opens up a whole new range of possibilities, due to it's large available storage memory.



How to use:

Configure the Cassiopei to be used on a VIC-20 computer, this is required in order to define all timings of the Cassiopei for this type of computer, see the chapter “Configuring the Cassiopei”. Do not attempt to use the Cassiopei while it is not configured.

The Cassiopei's requires the following sequence of commands independent of it's operating mode.

LOAD <return>

or

shift + RUN/STOP (this is even better because it saves the user the trouble of typing RUN later)

The computer will now ask you to “press play on tape”:

- press the PLAY/REC button on the Cassiopei to load the currently selected game/program
- press the MENU button in order to load the menu program to define your new settings.

The Cassiopei menu program is to be used for selecting the file (when usage of the Cassiopei manager is not possible or desired). Navigating through the menu is possible using the keyboard or by moving the joystick up/down and fire to select.

When a program is loaded using the Cassiopei's fastloader, the program is executed directly after it has finished loading. However .TAP files do not use the Cassiopei's fastloader and depending on the .TAP file used it may be required to type in RUN.

Note:

Do not press any key on the VIC-20's keyboard while loading as the keyboard of the VIC-20 shares IO-lines with the Cassetteport.

Memory expansion modules may be used but require some knowledge of the VIC-20's memory layout, for more info see the next page.

Important note about loading address:

The VIC-20 can have many different memory configurations. And these configurations affect the value of the BASIC start address. In order for an expanded VIC-20 to use BASIC-programs that are written for an un-expanded VIC-20, some relocation is required. This is less difficult then it seems. Because the Cassiopei can handle this fully automatic. The menu program will indicate the different situation with a color code at the bottom of the menu screen.

Green: *(this should work perfectly)*

the file matches the BASIC start address of the VIC-20's memory configuration

Blue: *(this may work if it is a pure BASIC prog without pokes to screen memory)*

the file does not match the BASIC start address of the VIC-20's memory configuration.

using BASIC address: file is a BASIC file, attempting relocation to current BASIC start

using file address: the file is an assembly program, no relocation, loading to file address

Red: *(the file is too large for the current BASIC memory configuration)*

the filesize is larger then the available BASIC memory, the file doesn't fit! This doesn't need to be a problem for certain machine language programs, for instance cracked games that were intended as cartridge games and require to be executed from cartridge memory area.

Normally the VIC-20 would relocate these basic programs by giving some extra information in the LOAD command. For instance LOAD",1,1 or LOAD",1,0 (more about that further on in this section). This however is a feature which can be that the Cassiopei can only use when it is in the slow loading modes. The Cassiopei cannot use this functionality when in the fastloader mode. Fortunately this has been compensated for in the fastloader itself, so all you need to do is type LOAD and nothing more. It can't be easier. This is possible because the Cassiopei can detect if the file to be loaded is a BASIC program or not. When the Cassiopei detect that the file to be loaded is a BASIC file, it will load the file to the current BASIC start location.

Now how does the Cassiopei know if a file is a BASIC or a machine code program. Well, BASIC programs on a VIC-20 will load to address \$0401, \$1001 or \$1201, therefore if the address in the file has one of these 3 addresses then that must be a BASIC file and relocation to the current BASIC start of the VIC-20 might be required.

When the address in the file is a different address then one of the above, the file is considered to be non-BASIC and will be loaded to the address as specified in the file. Therefore if you want to write your own machine language (a.k.a. assembly) program, then make sure that it does not start at one of the addresses as mentioned above, otherwise it will be treated as a BASIC program and might therefore be loaded to an unexpected memory location. Unless you implement a relocation algorithm as shown on the next pages. Which would be the best thing to do, because then the user can start a program with a RUN command instead of a SYS command, however this is only possible for relatively small programs. An example of such a small program are the Cassipei

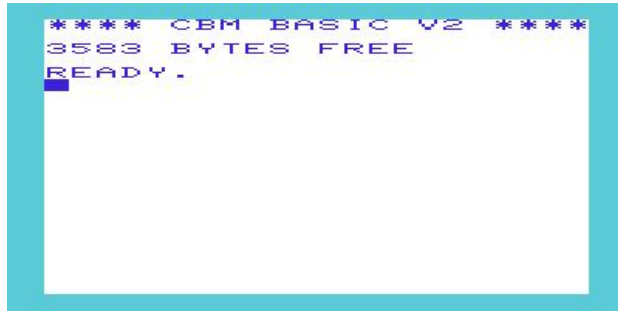
fastloader and the Cassiopei menu program, which works on all memory configurations of the VIC-20 and can be started with a simple RUN command.

What to do when a VIC-20 program does not run?

Make sure that you are using the right memory configuration. It might be possible that you are trying to load a file larger than the memory of your VIC-20, in that case add some additional memory. But it may also be the case that the program you want to load might be a simple machine language program with a BASIC stub. In those cases it is best to remove/disable the memory expansion and load the file only when the BASIC start address matches the start address of the file. Some memory expansion do not allow you to disable the 3K expansion that causes the BASIC start address to appear at \$0401, in order to disable that expansion (instead of removing the cartridge) a simple set of pokes is enough to achieve this: POKE642,16:POKE644,30:SYS64818

A problem with loading files are most likely to be caused by an incorrect setup of the system. A file that is too large to fit into the VIC-20's memory (because the file needs to be loaded on an expanded VIC-20 instead of an un-expanded one), then it cannot work. In case of doubt use an emulator with the same settings as your own VIC-20 in order to test if a program is capable of running on your current VIC-20 memory configuration (see the next page).

The VIC-20 has so many different configurations that it is a little confusing. Below an overview of different memory configurations and how they look like. The only directly visible difference is the available memory for BASIC. But what you can't see is that the BASIC start address and the video (screen and color) memory locations. Using the slow kernal loader, BASIC programs can be easily relocated. But machine language programs require some trickery to make it work. But a clever programmer can do this using his own relocation routines. An example is shown further on in this manual. Be aware of the changing Screen memory locations, this may also cause a relocated program to fail.



VIC-20: unexpanded
BASIC:\$1001, Screen:\$1E00, Color:\$9600



VIC-20: 3K expanded
BASIC: \$0401, Screen:\$1E00, Color:\$9600
(can be disabled by typing: POKE642,16:POKE644,30:SYS64818)



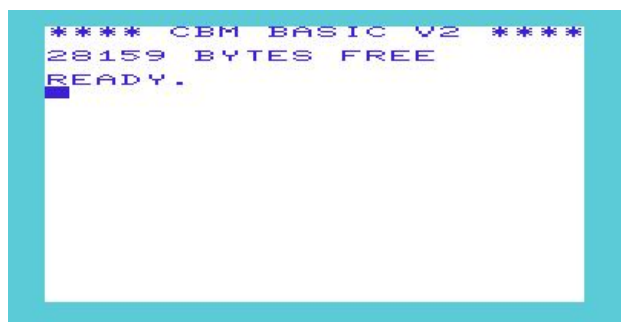
VIC-20: 3K super expander
BASIC: \$0401, Screen:\$1E00, Color:\$9600



VIC-20: 8K expanded
BASIC:\$1201, Screen:\$1000, Color:\$9400



VIC-20: 16K expanded
BASIC:\$1201, Screen:\$1000, Color:\$9400



VIC-20: 24K expanded
BASIC:\$1201, Screen:\$1000, Color:\$9400

Using the standard loader (slow speed) you can load a file in 2 different way's.

Using LOAD load the program to the BASIC start address.

Using LOAD""",1,1 load the program to the start address as specified in the file.

When you program a BASIC program on a +3K expanded VIC-20 and want to load that on an unexpanded VIC-20 then you will get into trouble if you load with LOAD""",1,1. Because then you will load your program into \$1001 (the BASIC start address of an unexpanded VIC-20). The +3K expanded VIC-20 has BASIC start \$0401, so this will not work, because when loading is ready you see nothing when you type list. This is because the BASIC program is not loaded into the proper basic area for the unexpanded machine. Fortunately the good people of Commodore have made an option to compensate for this. Simply load your file using LOAD and it will relocate automatically. This means that during loading all the addresses are correct for the current BASIC start value.

Relocating a BASIC program is possible, as long as it fits into the machine. Relocating a machine language program (not intended to be relocated) is not good. Because the address would change and the program would crash on execution, unless the program has it's own relocation routines.

When loading a file with the Cassiopei's fastloader, you cannot choose how to load the file. But this is not required. The Cassiopei will detect if the file will start at \$0401, \$1001 or \$1201 and if so, it must be a BASIC program and so it will be relocated to the current basic start address. If the program is a machine language program it will not start at one of these 3 locations. If the program is a hybrid program (a machine language program that begins with a BASIC stub then it will be relocated. But for these programs you must be aware if these programs are capable of running on your machine's configuration. Because not all programs will function. This is due to system constraints (or bad programming) and not because of the Cassiopei. However for most of these kind of games it is noted in a readme file or manual on what kind of memory configuration it should be played.

Technically it is possible to write a small program that has a BASIC stub and a self relocating machine language part. The Cassiopei menu program is such a program and below is shown how it works.

Example of a self relocating machine language program and a basic stub to start it:

The program below is written to work on memory address \$12XX and further, because this location is available in all possible memory configurations. When compiled using the address below, it can be loaded and started with a simple run. No more SYS commands to remember, just RUN. The relocating routine will copy itself from the current location the correct location, but because these areas might overlap, the copying routine copies the last byte first. In other words, it copies backwards, so the loop ends by copying the first byte last.

```
;zero page variables
;=====
SOURCE_ADR      = $F7      ;pointer for relocation
;SOURCE_ADR+1    = $F8      ;
DEST_ADR         = $F9      ;pointer for relocation
;DEST_ADR+1      = $FA      ;

;BASIC stub
;=====
*=$1201
        ;20 SYS32+256*peek(44)

        ;Start address $1201, end address $1214
        BYTE $13,$10,$14,$00,$9e,$33,$32,$aa,$32,$35,$36,$ac,$c2,$28,$34,$34,$29,$00,$00,$00

;Code relocater (this code can run from any address)
;=====
*=$1220
```

```

RELOCATE      LDA $2C          ;
               CMP #$12        ;
               BEQ RELOCATE_DONE ;code already at the correct location
               ;program_size = END_OF_PROGRAM - INIT
               ;first byte in memory = contents of $2C + low byte of label MAIN
               ;last byte in memory = first byte in memory + program_sizeRELOCATE_REQ
               LDA #<END_OF_PROGRAM ;store destination address in zero-page variable
               STA DEST_ADR        ;=====
               LDA #>END_OF_PROGRAM ;
               ;
               STA DEST_ADR+1      ;high byte
               LDA #<MAIN          ;calculate first byte of code to be relocated in memory
               STA SOURCE_ADR      ;=====
               LDA $2C            ;
               STA SOURCE_ADR+1    ;high byte
               ;-----
               ;ATTENTION: make sure that the option "calc address first then high/low byte"
               ;in CBM program studio is selected. Otherwise the lines below WILL fail!!!!!!
               ;-----
               CLC                  ;add program size to get to the last byte
               LDA SOURCE_ADR       ;=====
               ADC #<END_OF_PROGRAM-MAIN ;carry that we be used in the next addition
               STA SOURCE_ADR       ;
               LDA #>END_OF_PROGRAM-MAIN ;
               ADC SOURCE_ADR+1      ;add carry to high byte (if there was any)
               STA SOURCE_ADR+1     ;
               CLV                  ;clear flag here to keep the loop fast
               LDY #$00             ;clear reg here keeps the loop fast
RELOCATE_LP    LDA (SOURCE_ADR),Y    ;the actual moving of the program (the copy loop)
               STA (DEST_ADR),Y     ;=====
               LDA DEST_ADR+1       ;check if last byte reached
               CMP #>MAIN           ;=====
               BNE RELOCATE_00      ;
               LDA DEST_ADR         ;low byte of last written address
               CMP #<MAIN           ;
               BEQ RELOCATE_DONE    ;
RELOCATE_00    DEC SOURCE_ADR       ;calculate next addresses
               LDA SOURCE_ADR       ;
               CMP #$FF             ;
               BNE RELOCATE_01      ;=====
               DEC SOURCE_ADR+1     ;overflow detected
RELOCATE_01    DEC DEST_ADR         ;decrement low-byte
               LDA DEST_ADR         ;
               CMP #$FF             ;
               BNE RELOCATE_02      ;check for overflow of low byte
               DEC DEST_ADR+1       ;overflow detected, so we must also decrement high-byte
RELOCATE_02    ;CLV                 ;force conditional branch because
               BVC RELOCATE_LP      ;absolute jumps are impossible
RELOCATE_DONE JMP MAIN             ;code is at correct location, so we may start it

;Main program (the code that will be relocated), it may not exceed $1DFF
;This means that your own program may be bigger then approximately 3Kbyte
;=====

MAIN
;;;write your own program here;;;

END_OF_PROGRAM ;this label indicates the absolute end of the program

```

2.4.4 Commodore C16/Plus4

The C16/Plus4 computers are slightly different due to the fact that their cassetteport uses a slightly different connector. Fortunately this can be easily solved by using the CBM's standard connector adapter.



How to use:

Configure the Cassiopei to be used on a C16/Plus4 computer, this is required in order to define all timings of the Cassiopei for this type of computer, see the chapter “Configuring the Cassiopei”. Do not attempt to use the Cassiopei while it is not configured.

The Cassiopei's requires the following sequence of commands independent of it's operating mode.

LOAD <return>

The computer will now ask you to “press play on tape”:

- press the PLAY/REC button on the Cassiopei to load the currently selected game/program
- press the MENU button in order to load the menu program to define your new settings.

The Cassiopei menu program is to be used for selecting the file (when usage of the Cassiopei manager is not possible or desired). Navigating through the menu is possible using the keyboard or by moving the joystick (port 1 or 2) up/down and fire to select.

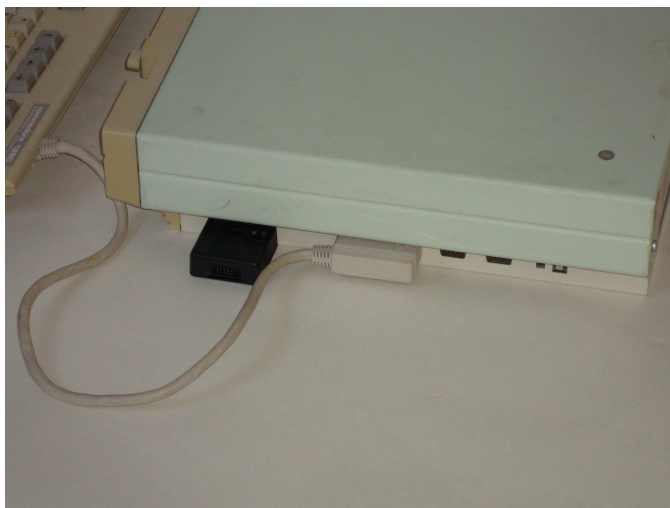
When a program is loaded using the Cassiopei's fastloader, the program is executed directly after it has finished loading. However .TAP files do no use the Cassiopei's fastloader and depending on the .TAP file used it may be required to type in RUN.

2.4.5 Commodore 128

The C128 is a great computer with many features. The C128 models as shown in the pictures below have an internal diskdrive that can make the use of the Cassiopei even more fun.

The Cassiopei (for use on a C128) must be configured for a C128. When a game or program for use with a C64 is selected using the Cassiopei menu-program it will automatically switch the C128 into C64 mode before it starts the selected game/program. How does this work you might ask, well when you exit the menu-program the computer orders the Cassiopei to briefly go into the C64 mode and also the C128 computer is switched into C64 mode. Then the selected game is loaded and after that the Cassiopei will return to C128 mode.

This way you never have to start your C128 with a cartridge installed (to force it into C64 mode) or press the CBM key during reset/power-up. No, you can allways switch on you C128 in C128 mode, just the way the good people of Commodore intended it to.



C128D "metal model"



C128D "plastic model"

Although it is OK to use the C128 only in C64 mode (if you do, please see the section C64 in this manual) there really is no need to. Even if you are using the C128 only for playing C64 games, then it still would bennefit if you boot up in C128 mode. As it offers a great function that may save you some typing time... every time you switch on the computer. The trick is in the C128 bootdisk functionality. When a bootdisk is installed, the C128 will load a program from that disk and executes it. Fully automatic. The Cassiopei menu-program has a function to create such a bootdisk. If you use this function to make such a disk, then when you hav the disk inserted at the moment you switch on your C128, the Cassiopei menu-program will automatically load and within seconds you'll see the Cassiopei menu program. And you can now select your favorite game or program. It would "feel" like you are working from a cartridge (slightly slower, but a lot more flexible).

Dual screen, the Cassiopei menu-program allows the use of the C128 in 40 or 80 column mode. So no matter in which mode you work, you can use the same menu-program without any problems.

How to use:

Configure the Cassiopei to be used on a C128 computer, this is required in order to define all timings of the Cassiopei for this type of computer, see the chapter “Configuring the Cassiopei”. Do not attempt to use the Cassiopei while it is not configured.

The Cassiopei's requires the following sequence of commands independent of it's operating mode.

LOAD <return>

The computer will now ask you to “press play on tape”:

- press the PLAY/REC button on the Cassiopei to load the currently selected game/program
- press the MENU button in order to load the menu program to define your new settings.

Now the Commodore 128 starts looking for a program. When the program is found, the Commodore 128 prints FOUND "filename", the filename is the name of the selected program or the fastloader required to load the selected program or menu. Press the COMMODORE-key to LOAD the this file or wait a few seconds and loading will continue. **However if you press the spacebar the file will be skipped and loading eventually stops. This behavior is different from the behavior of that of other Commodore machines (like for instance on the C64). This different behaviour is embedded in the design of the C128 and is not caused in any way by the Cassiopei.**

The Cassiopei menu program is to be used for selecting the file (when usage of the Cassiopei manager is not possible or desired). Navigating through the menu is possible using the keyboard or by moving the joystick (port 1 or 2) up/down and fire to select. When you press the “return” key (or move the joystick left/right) you may switch between C64 and C128 mode. Only the files marked for that type of system will be shown.

When a program is loaded using the Cassiopei's fastloader, the program is executed directly after it has finished loading. However .TAP files do no use the Cassiopei's fastloader and depending on the .TAP file used it may be required to type in RUN.

Note:

On all systems it is possible to directly start the last selected game/program (without use of the menu) by typing LOAD and pressing the play-button on the Cassiopei. However the C128 is a special case...

You can start the last selected C128 game/program on a C128 in C128 mode, by typing LOAD and pressing the play button on the C128. But you cannot start a C64 game (on a C128 in C128 mode) by typing LOAD and pressing the play-button on the Cassiopei. You need to go into the Cassiopei menu to start a C64 game on a C128 in C128 mode.

The only alternative is to start the C128 in C64 mode and to use the Cassiopei configured specifically for C64 use. But that would eliminate all benefits the C128 has to offer and is therefore not recommended.

2.4.6 Commodore PET/CBM 20XX series (build-in tape drive)

Commodore made a name in the computer industry by the iconic PET computer. With its adorable (but not so practical) keyboard and the build-in tape recorder this machine made low cost computing possible and started a revolution in the industry. Commodore made this model in different memory configurations (8, 16, 32K). Despite from memory differences, these models all work the same and are therefore referred to as the 20XX series in this manual. The 20XX series are well known for their build in cassette player (cass#1) but it has also an cassetteport #2. Cassetteport#1 is located inside and connected to the build-in tape drive. Therefore it is not as easily accessible as cassetteport#2, which is conveniently located at the back of the computer. Theoretically the Cassiopei could function on both cassetteports, but the speedloader software inside the Cassiopei was designed for this model to operate using cassetteport#2 (because this is easily accessible AND to allow the use of both cassettebuffers for the fastloader routines). Therefore it will not fully work when connected to the internal cassetteport#1. Unfortunately, this PET model was equipped with a buggy version BASIC. Which results in some issues when working with the Cassiopei, however, these issues can be worked around or ignored.



Usage:

Configure the Cassiopei to be used on a PET/CBM 20XX (build-in tape drive) computer, this is required in order to define all timings of the Cassiopei for this type of computer, see the chapter "Configuring the Cassiopei". Do not attempt to use the Cassiopei while it is not configured.

The Cassiopei's requires the following sequence of commands independent of its operating mode. Be aware there are differences between the various versions of Commodore BASIC.

LOAD"";2	(BASIC 1, startup screen shows *** COMMODORE BASIC ***)
LOAD"";2	(BASIC 2, startup screen shows ### COMMODORE BASIC ###)
LOAD"";2	(BASIC 4, startup screen shows *** COMMODORE BASIC 4.0 ***)

The computer will now ask you to "press play on tape":

- press the PLAY/REC button on the Cassiopei to load the currently selected game/program
- press the MENU button in order to load the menu program to define your new settings.
- depending on method of loading type **RUN** to start the menu program

Note to BASIC 1 users:

BASIC 1 is the first version of BASIC that Commodore released on their PET computers, unfortunately, this BASIC version has many problems and Commodore has quickly responded to this by the release of an upgraded BASIC version, BASIC 2. It is preferred that machines with BASIC 1 are upgraded to BASIC 2. Therefore the use of the Cassiopei in combination with a computer using BASIC 1 might result in unexpected messages after loading and running. Sometimes an error in a BASIC line is reported, but after typing list, the program will run without problems. The easiest way to prevent these strange problems is to use the slow kernal loader (use the mode named “slow”, in the menu program).

Note:

The Cassiopei uses a part of the stack and the cassettebuffer#1 to store the fastloader routines. Some programs are loaded into the area of tape buffer#2, although the Cassiopei is connected to cassette-port 2 this does not interfere with the fastloader routines. This because the fastloader moves itself into tape buffer#1 freeing tape buffer#2 for programs.

Some programs cannot be loaded using the Cassiopei fastloader. For instance the game “Wasps” which loads to memory locations \$027A - \$1400. (this information is visible in the Cassiopei manager). This uses tape buffer#1 (which starts at \$027A). However it is still possible to load this game using the Cassiopei, but it requires the kernal loader. By loading the game using the standard kernal loader (also known as the slowest method of loading) the game is loaded exactly as it was designed in the early 80's.

2.4.7 Commodore PET/CBM 20XX and 30XX series

Commodore has build many variations of the PET 2000 series. Not only did they vary the amount of RAM, they also removed the build-in tape drive. Which is a bit of an issue because the cassette port numbers are different between the different model of the same serie. However this is easily noticeable. Because the build-in tape drive is always #1 and the external tape drive is #2, if there is no build-in tape drive then the external tape drive is called #1. The Cassiopei fastloader software needs to know where the tape drive is located (#1 or #2) therefore the proper computermodel must be chosen in the settings screen.

Soon after the release of the 2000 series Commodore came with the 3000 series, which is very similar to the 2000 series that have no build-in tape drive. In fact they are so similar that the Cassiopei can use the same fastloader routines for both series of machines. Regarding the 3000 series, Commodore made this model also in different memory configurations (3008, 3016, 3032). Despite from memory differences, these models all work the same and are therefore referred to as the 30XX series in this manual.



Useage:

Configure the Cassiopei to be used on a PET/CBM 20XX & 30XX computer, this is required in order to define all timings of the Cassiopei for this type of computer, see the chapter “Configuring the Cassiopei”. Do not attempt to use the Cassiopei while it is not configured.

The Cassiopei's requires the following sequence of commands independent of it's operating mode. Be aware the there are differences between the various versions of Commodore BASIC.

<not supported>	(BASIC 1, startup screen shows *** COMMODORE BASIC ***)
Shift + RUNSTOP	(BASIC 2, startup screen shows ### COMMODORE BASIC ###)
<see note>	(BASIC 4, startup screen shows *** COMMODORE BASIC 4.0 ***)

Note: Please note that a 30XX with BASIC 4 functions like a 40XX, so in this case please configure the Cassiopei for use with a 40XX and use the 40XX menu program.

The computer will now ask you to “press play on tape”:

- press the PLAY/REC button on the Cassiopei to load the currently selected game/program
- press the MENU button in order to load the menu program to define your new settings.
- depending on method of loading type **RUN** to start the menu program

Note:

When a program is loaded using the Cassiopei's fastloader, the program is executed directly after it has finished loading. However .TAP files do not use the Cassiopei's fastloader and depending on the .TAP file itself it may be required to type in RUN.

Some programs cannot be loaded using the Cassiopei fastloader. For instance the game “Wasps” which loads to memory locations \$027A - \$1400. (this information is visible in the Cassiopei manager). This uses tape buffer#1 (which starts at \$027A) which is also used by the fastloader routines of the Cassiopei. However it is still possible to load this game using the Cassiopei, but it requires the kernal loader. By loading the game using the standard kernal loader (also known as the slowest method of loading) the game is as it was designed in the early 80's.

In some occasions when the RUN command is entered, the Cassiopei loads the program, but the computer responds with an error message. Most of the times entering the RUN command again will start the loaded program without any problems. In those occasions it would have been better to use a SYS command to start the fastloader routines. So instead of RUN, type SYS 1040 (Also shown when typed list after loading of the fastloader program).

2.4.8 Commodore PET/CBM 40XX series

Commodore has build many variations of the PET, one of them is the PET 4032. Commodore made this model in different memory configurations (4008, 4016, 4032). Despite from memory differences, these models all work the same and are therefore referred to as the 40XX series in this manual. The 40XX series do not have a build in cassette player but it has 2 cassetteports. Cassetteport#1 is easily accessible while cassetteport#2 is located inside. Theoretically the Cassiopei could function on both cassetteports, but the speedloader software inside the Cassiopei was designed for use on cassetteport#1 (because this is easily accessible and allows the use of both cassettebuffers for storing the fastloader routines). Therefore it will not fully work on cassetteport#2. Also loading from this port allows the user to type **LOAD** without specifying the port address as cassetteport#1 is the default cassetteport.

Usage:

Configure the Cassiopei to be used on a PET/CBM 40XX computer, this is required in order to define all timings of the Cassiopei for this type of computer, see the chapter “Configuring the Cassiopei”. Do not attempt to use the Cassiopei while it is not configured.

The Cassiopei's requires the following sequence of commands independent of it's operating mode. Be aware the there are differences between the various versions of Commodore BASIC.

<see note>	(BASIC 1, startup screen shows *** COMMODORE BASIC ***)
<see note>	(BASIC 2, startup screen shows ### COMMODORE BASIC ###)
LOAD <return>	(BASIC 4, startup screen shows *** COMMODORE BASIC 4.0 ***)

Note: Basic version 1 and 2 were never released for the 40XX series, if your computer uses these versions, the system has been downgraded. This practice is not supported by the Cassiopei. For this and for historical reasons it is suggested to restore the computer to it's original state. So that it can function as intended by the nice people of Commodore.

The computer will now ask you to “press play on tape”:

- press the **PLAY/REC** button on the Cassiopei to load the currently selected game/program
- press the **MENU** button in order to load the menu program to define your new settings.
- depending on method of loading type **RUN** to start the menu program

Note:

When a program is loaded using the Cassiopei's fastloader, the program is executed directly after it has finished loading. However .TAP files do no use the Cassiopei's fastloader and depending on the .TAP file itself it may be required to type in **RUN**.

Some programs cannot be loaded using the Cassiopei fastloader. For instance the game “Wasps” which loads to memory locations \$027A - \$1400. (this information is visible in the Cassiopei manager) which is also used by the fastloader routines of the Cassiopei. This uses tape buffer#1 (which starts at \$027A). However it is still possible to load this game using the Cassiopei, but it requires the kernal loader. By loading the game using the standard kernal loader (also known as the slowest method of loading) the game is as it was designed in the early 80's.

In some occasions when the **RUN** command is entered, the Cassiopei loads the program, but the computer responds with an error message. Most of the times entering the **RUN** command again will

start the loaded program without any problems. In those occasions it would have been better to use a SYS command to start the fastloader routines. So instead of RUN, type SYS 1040 (Also shown when typed list after loading of the fastloader program).

2.4.9 Commodore PET/CBM 80XX series

Commodore has build many variations of the PET, one of them is the PET 8032. Commodore made this model in different memory configurations (8008, 8016, 8032). Despite from memory differences, these models all work the same and are therefore referred to as the 80XX series in this manual. The 80XX series do not have a build in cassette player but it has 2 cassetteports. Cassetteport#1 is easily accessible while cassetteport#2 next to the memory expansion connector. Theoretically the Cassiopei could function on both cassetteports, but the speedloader software inside the Cassiopei was designed for use on cassetteport#1 (because this is easily accessible and allows the use of both cassettebuffers for storing the fastloader routines). Therefore it will not fully work on cassetteport#2. Also loading from this port allows the user to type LOAD without specifying the port address as cassetteport#1 is the default cassetteport.



Useage:

Configure the Cassiopei to be used on a PET/CBM 80XX computer, this is required in order to define all timings of the Cassiopei for this type of computer, see the chapter “Configuring the Cassiopei”. Do not attempt to use the Cassiopei while it is not configured.

The Cassiopei's requires the following sequence of commands independent of it's operating mode. Be aware the there are differences between the various versions of Commodore BASIC.

<see note>	(BASIC 1, startup screen shows *** COMMODORE BASIC ***)
<see note>	(BASIC 2, startup screen shows ### COMMODORE BASIC ###)
LOAD <return>	(BASIC 4, startup screen shows *** COMMODORE BASIC 4.0 ***)

Note: Basic version 1 and 2 were never released for the 40XX series, if your computer uses these versions, the system has been downgraded. This practice is not supported by the Cassiopei. I would kindly advise to restore the computer to it's original state. So that it can function as intended by the nice people of Commodore.

The computer will now ask you to “press play on tape”:

- press the PLAY/REC button on the Cassiopei to load the currently selected game/program
- press the MENU button in order to load the menu program to define your new settings.
- depending on method of loading type **RUN** to start the menu program

Note:

When a program is loaded using the Cassiopei's fastloader, the program is executed directly after it has finished loading. However .TAP files do not use the Cassiopei's fastloader and depending on the .TAP file itself it may be required to type in RUN.

Some programs cannot be loaded using the Cassiopei fastloader. For instance the game “Wasps” which loads to memory locations \$027A - \$1400. (this information is visible in the Cassiopei manager). This uses tape buffer#1 (which starts at \$027A) which is also used by the fastloader routines of the Cassiopei. However it is still possible to load this game using the Cassiopei, but it requires the kernal loader. By loading the game using the standard kernal loader (also known as the slowest method of loading) the game is as it was designed in the early 80's.

In some occasions when the RUN command is entered, the Cassiopei loads the program, but the computer responds with an error message. Most of the times entering the RUN command again will start the loaded program without any problems. In those occasions it would have been better to use a SYS command to start the fastloader routines. So instead of RUN, type SYS 1040 (Also shown when typed list after loading of the fastloader program).

2.4.10 Commodore CBM 600 series (not supported)

Support for the Commodore 610, shown below, is impossible.



The picture above shows that it is possible to connect the Cassiopei to the CBM 600 series, unfortunately that's about it. You cannot give the command to load, rendering the Cassiopei useless.

It appears that Commodore has removed the routines that access the tape. There were several sets of ROMs around for these machines. Early ones apparently contained the code to run the cassette port though this was dropped to make room for the IPC code to interface with the second processor. Any attempt to load or save to tape on the later versions gives the error '?DEVICE NOT PRESENT'.

This means that the Cassiopei would require a cartridge or diskdrive with special software to start it. This however was not the intention of the Cassiopei's design. As the Cassiopei was designed to be a device that would not require other storage devices in order to work. So technically it is possible to use the Cassiopei in combination with the CBM 600 series, but it requires a disk-drive or cartridge as well. With this in mind I (Jan Derogee, inventor of the Cassiopei) decided that support for this computer model is not really useful.

2.5 General usage of the Cassiopei

The following paragraphs will describe the overall functionality of the Cassiopei, functionality that is similar for all supported computer models.

2.5.1 Compatibility

Cassiopei fastloader

Some programs (and these are very rare) will not load using the Cassiopei fastloader for a very simple reason. If the program that must be loaded needs uses the same memory space as the Cassiopei's fastloader routines then the program that is loaded overwrites the fastloader program and the loading simply stops. The only solution to this problem would be to load the program using the "standard kernel loader" (a.k.a. the "slow" method of loading).

For the VIC-20, C64, C16/+4 the cassette buffer is also used for storing the fastloader code, these computers do not require the use of the stack to store the fastloader code.

For the PET the fastloader stores itself into the cassettebuffer (Cassette buffer#1 and a small portion of the stack). The Cassiopei uses a portion of the stack because the fastloader needs to be stored somewhere and Cassettebuffer#2 would be a nice place BUT this is exactly what the earlier programmers thought as well, so therefore cassette buffer #2 can not be used by the Cassiopei fastloader code, fortunately the stack is large enough to store a part of the fastloader code. The reason why the PET's fastloader is larger then the C64, VIC-20, C16/+4 fastloader code is simply because the IO-lines are much easier to control on the non-PET computers and therefore requires less code.

Jiffy-DOS

The Cassiopei is not compatible with JiffyDOS equipped computers. This because Jiffy-DOS disables the use of the cassette and occupies the cassette buffer in memory. Therefore the Cassiopei cannot function properly. A simple workaround would be to disable JiffyDOS or to load the programs that require the use of the Cassiopei from disk.

Cartridges

The Cassiopei may function with various kind of cartridges, however cartridges that add functionality in the form of speedloaders and BASIC extensions might cause problems as they might occupy important memory locations (BASIC jump vectors). Regarding the "PRG slow loading mode" (standard kernal loader) and the ".TAP file loading mode" there are no incompatibilities to be expected (considering the .TAP and .PRG program itself is compatible with the cartridge).

The Chameleon cartridge, on the C64, is a completely different story, this cartridge takes over the C64's processor. And because the cassetteport is connected to the C64's processor the tape functionality simply does no longer work!

Cassetteport related hardware

The Cassiopei should not be used in combination with other hardware connected to the cassetteport. For instance, the 1541 ultimate tape adapter or a cassetteport splitter allowing to connect more then 1 datasette to the cassetteport. This because the Cassiopei uses the IO-lines of the cassetteport in a more advanced way then then all other existing hardware. When combining other cassetteport hardware with the Cassiopei damage might occur.

2.5.2 LOAD and SAVE

The Cassiopei can LOAD and SAVE using the standard tape protocol. This means that (if your Cassiopei is configured correctly) you can type in a program on your CBM computer and type the SAVE"<FILENAME>" command. When the computer asks you to "PRESS REC AND PLAY ON TAPE", you press the Play/Rec button on the Cassiopei and the computer will record the program. The program will be stored onto the Cassiopei's flash memory and can be copied to your PC using USB and the Cassiopei manager software.

Attention:

Do not press the Play/Rec button before the computer says "PRESS REC AND PLAY ON TAPE" because if you do the Cassiopei cannot detect the recording signal and will go into play mode. If you accidentally press too soon, simply press the reset button on the Cassiopei, wait 2 seconds and then do what you intended to do.

Once you are saving using the SAVE command you'll notice that the Cassiopei stops at exactly half the time it takes to save the file. This is correct and for a good reason. Commodore's tape protocol always sends the file twice. Meaning that the file you save will be stored twice on your tape. This is called redundancy and is used for protecting the file against errors on the tape. But because the Cassiopei's flash memory is much more reliable than the tape system, there is no need to use this redundancy and the Cassiopei saves the file to the flash memory and ignores the redundant data. Saving you half the time. Once you've noticed that the Cassiopei is ready with saving the file (LED activity changes to indicate this), you may abort the saving operation using the run/stop button of your Commodore.

Note: when saving a file make sure that you choose a unique filename for your file to be saved. If you choose a filename that already exists on the Cassiopei's flash memory then you have files with identical names, no data is lost but it will become very confusing for the user to choose the correct file. Renaming files on the Cassiopei's flash memory filesystem is not possible. If you want to rename a file, then extract the file to your PC, delete it from the Cassiopei's flash, rename it on the PC and copy it to the Cassiopei.

Note: the Cassiopei can store up to 8MByte of data on its internal flash memory. However, considering the time it takes to save using the standard tape protocol, the max. file size of a file to save using the SAVE command is limited to a max. filesize of 128KByte. Which is a lot and in most cases more than the memory of the CBM computer you are using. Food for thought: the standard tape protocol has an effective transfer rate of 50 bytes/sec, meaning that you'll be waiting for more than 43 minutes before you've saved more than the buffer can handle (>128KBytes).

2.5.3 TAP file usage

.TAP files are images of real cassette tapes. That's the main reason why .TAP files are so big. Each and every signal of the tape is very accurately described in the .TAP file. .TAP files are not to be compared with .PRG files as .PRG files are programs directly interpretable while .TAP files are small programs and lot's of data. The only way to interpret the data is by loading and executing the loader at the beginning of the .TAP file. This is one of the reasons why .TAP files cannot be loaded faster then the speed they were designed for (which is according to today's standard pretty slow...)

.TAP files are to be loaded by selecting the .TAP file mode in combination with the selection of a .TAP file using the Cassiopei's menu program or the Cassiopei manager. The Cassiopei's buttons in TAP file mode have the following functionality:

Play/Rec: start/resume loading of the TAP file from the TAP-file position. The current position of a TAP file is 0 after reset of the Cassiopei. The LED shall now light green or blinking (blinking indicates that the play mode is active but the CBM computer does not drive the motor. In some situation this means that you need to pres the CBM key, in other situations it is an indicating that loading has finished. When color RED is visible it indicates USB connectivity.

Menu: stop the playback of the .TAP file. The LED is no longer green. Depending on USB connectivity is shows the color RED

Reset: stop playback and set the .TAP file to the 0-position.

As you might have noticed, there is no method of forwarding or rewinding the TAP file using the Cassiopei's buttons. If this functionalty is required, then the Cassiopei manager offers a solution. The Cassiopei manager's has a TAP-tab, select this tab and you'll notice a commodore datasette. That datasette represents the Cassiopei's TAP functionality. Now you can (by pressing the corresponding buttons) FastForward or Rewind with the aid of a tape counter. Just like a real datasette. Even the sounds of the datasette are functional, this is done to create a more realistic tape experience. You may choose to press the small buttons on the datasette or the larger buttons on the right side of the screen

The status button can be used to monitor the tape counter for the situation where a TAP file is started without the use of the Cassiopei manager.

The counter shows the current byte being processed by the Cassiopei. By showing bytes instead of time

A more accurate position inside the TAP file can be shown. Showing a tape counter that resembles the original datasette tape counter (based on time) is very difficult. Mostly because tape counters differ very much between the various kinds of models of tape players. So there would never be an implementation of a tapecounter that behaves exactly like every all tape player ever produced. Although an approximation could be achieved, it would result in a lot of programming difficulties that can be easily avoided by basing the counter on the number of bytes that are read from the .TAP file. And that is exactly what the Cassiopei manager does.

Swapping or flipping the tape (a.k.a. changing the TAP during playback)

When you are in a TAP-file game and you must change to a different TAP file, simply because TAP images of games are split up into 2 different TAP files. A TAP-file for side A of the tape and a TAP-file for side B of the tape. Or perhaps you must insert a data cassette, because your application uses a TAP for the program and a TAP for the data. Although writing to TAP files is not possible for various reasons.

When you are playing back the TAP files from flash. Then you must stop the playback by pressing the reset button of the Cassiopei (wait 3 seconds before doing anything else). Then using the Cassiopei manager you choose the new file by selecting the new TAP file in the File screen and you click with your right mouse button on the selected file, you will now see a menu appear that shows the item "use this file" and you click this on item. Now the Cassiopei is configured to playback this TAP file from flash. So when you press the play button of the Cassiopei, the TAP file is played. You may start this TAP file by pressing the play button in the TAP screen. You may navigate through this file using the forward and reverse buttons in the TAP screen.

When you are using the virtual TAP file mode then you must stop the TAP file playback (pressing the stop button in the cassiopei manager (TAP screen). Go to the virtual tab screen and using the button "select file" you select the new TAP file. Then you press the "transfer virtual TAP file" and the new TAP file is played. You may navigate through this file using the forward and reverse buttons in the TAP screen.

2.5.4 Cross development

The Cassiopei is a great device to use for cross development as you can very quickly transfer data between your PC development suite to your CBM computer. Sure, a lot of testing can be done using emulators, but sometimes you just require the real thing for proper testing.

A great tool for the actual programming would be CBM program studio from Arthur Jordison (<http://www.ajordison.co.uk/index.html>), this tool allows you to program in BASIC and Assembly language for a broad range of platforms (PET 2001, VIC-20, C64, C128, C16/Plus4). All example programs for the Cassiopei were written using this tool.

The best way to learn programming, is by taking an existing program and taking it apart... modifying it until it does what you want it to do. And now you have the all tools for it! For the fastest method of testing, use the virtual file mode. With a minimum of key presses your program is transferred into your CBM computer.

2.5.5 BASIC wedge (C64)

Attention:

This manual assumes that you have the menu program installed that incorporates the BASIC wedge, this because there is also a menu program available that does not include the wedge (this because of it's smaller size will load much faster).

The BASIC wedge is a small program that adds extra BASIC commands to the standard BASIC. The wedge is stored in a safe location that is not being used by BASIC routines. You can load the wedge by loading the menu program (simply press the menu button after giving the load command and the computer requests you to “press play on tape”). When the menu program (or menu) is loaded you can exit without choosing a file using the wedge function, then the wedge appears and you can try it out. The idea by putting the wedge in the menu program is simple. When you load your programs through this tool, the wedge does not need to be loaded separately. The wedge remains in memory until the computer is switched off OR it is overwritten by another program. You can test the presence of the wedge by typing the following command:

```
!HELP
```

If you want to load a program that uses the wedge, just select that program in the menu and exit using the “save and exit” function, the program you've chosen is now loaded. If your program is not overwriting the wedge then it will be available during the run time of your loaded program. However in order to use it it must be enabled first! The example below, activates the wedge in the first line of the program

```
5 SYS 49500 :REM ACTIVATE WEDGE (C64)
10 A=!ADC,0 :REM ADC VALUE CHAN-0
20 PRINT"ADC VALUE =";A
30 GOTO 100
```

Although resetting your computer doesn't delete the wedge, it does require to be activated. It can be re-activated by using the SYS command as shown in the example above.

For many more examples about the Cassiopei's functionality and the wedge, see the chapter “CPIO messages definitions”

2.5.6 BASIC wedge (C128)

Attention:

This manual assumes that you have the menu program installed that incorporates the BASIC wedge, this because there is also a menu program available that does not include the wedge (this because of it's smaller size will load much faster).

The BASIC wedge is a small program that adds extra BASIC commands to the standard BASIC. The wedge is stored in a safe location that is not being used by BASIC routines. You can load the wedge by loading the menu program (simply press the menu button after giving the load command and the computer requests you to “press play on tape”). When the menu program (or menu) is loaded you can exit without choosing a file using the wedge function, then the wedge appears and you can try it out. The idea by putting the wedge in the menu program is simple. When you load your programs through this tool, the wedge does not need to be loaded separately. The wedge remains in memory until the computer is switched off OR it is overwritten by another program. You can test the presence of the wedge by typing the following command:

```
!HELP
```

If you want to load a program that uses the wedge, just select that program in the menu and exit using the “save and exit” function, the program you've chosen is now loaded. If your program is not overwriting the wedge then it will be available during the run time of your loaded program. However in order to use it it must be enabled first! The example below, activates the wedge in the first line of the program

```
5 SYS 14444 :REM ACTIVATE WEDGE (C128)
10 A=!ADC,0 :REM ADC VALUE CHAN-0
20 PRINT"ADC VALUE =";A
30 GOTO 100
```

Although resetting your computer doesn't delete the wedge, it does require to be activated. It can be re-activated by using the SYS command as shown in the example above.

For many more examples about the Cassiopei's functionality and the wedge, see the chapter “CPIO messages definitions”

2.5.7 BASIC wedge PET/CBM3000 series

Attention:

This manual assumes that you have the menu program installed that incorporates the BASIC wedge, this because there is also a menu program available that does not include the wedge (this because of it's smaller size will load much faster).

The BASIC wedge is a small program that adds extra BASIC commands to the standard BASIC. The wedge is stored in a safe location that is not being used by BASIC routines. You can load the wedge by loading the menu program (simply press the menu button after giving the load command and the computer requests you to "press play on tape"). When the menu program (or menu) is loaded you can exit without choosing a file using the wedge function, then the wedge appears and you can try it out. The idea by putting the wedge in the menu program is simple. When you load your programs through this tool, the wedge does not need to be loaded separately. The wedge remains in memory until the computer is switched off OR it is overwritten by another program. You can test the presence of the wedge by typing the following command:

```
10 !HELP
```

```
RUN
```

The example above is a simple program. And all wedge commands must be entered like a program. When you type: !HELP and press return, you will get a syntax error. This is because the BASIC wedge only allows indirect commands. Or in other words commands during program execution. This differs from the C64 which does allow the entering of wedge commands in direct mode. The C64 can do this because it has a slightly more advanced basic that makes it easier to create a wedge.

The PET/CBM 3000 series wedge also does not allow the have wedge commands that return a value into a variable. So it is not possible to use the all the commands as seen in the various examples in this document, below a table that shows the differences between the wedges. Using this table it should be possible to convert any example for the C64 to work on the PET/CBM 3000 series wedge.

WEDGE syntax differences		
C64	PET/CBM3000 series	Information
A=!ADC,0 A=!I2CGE A=!I2CGL	A=0:!ADC,0 A=0:!I2CGE A=0:!I2CGL	Read ADC value Read I2C value Read I2C value (last byte)
PRINT !ADC,0 PRINT !I2CGE PRINT !I2CGL	A=0:!ADC,0:PRINT A A=0:!I2CGE:PRINT A A=0:!I2CGL:PRINT A	Read and print ADC value Read and print I2C value Read and print I2C value (last byte)

What the wedge does is that it searches for the variable A (so do not use a different name) and then it sends the value of the wedge command result to that variable. **If you use any other variable then A, it will not work!!!**

If you want to load a program that uses the wedge, just select that program in the menu and exit using the “save and exit” function, the program you've chosen is now loaded. If your program is not overwriting the wedge then it will be available during the run time of your loaded program. However in order to use it it must be enabled first! The example below, activates the wedge in the first line of the program

```
5 SYS 29000 :REM ACTIVATE WEDGE (PET/CBM3000)
10 A=0:!ADC,0 :REM ADC VALUE CHAN-0
20 PRINT"ADC VALUE =" ;A
30 GOTO 100
```

Although resetting your computer doesn't delete the wedge, it does require to be activated. It can be re-activated by using the SYS command as shown in the example above.

For many more examples about the Cassiopei's functionality and the wedge, see the chapter “CPIO messages definitions”

2.5.8 BASIC wedge for all other models

The BASIC wedge has not been developed for other models then mentioned in the previous chapters. This is because the various forms of BASIC as released by Commodore and the available memory for the different types of computers as produced by Commodore varies too much to implement all models at this moment. This due to the low priority of the BASIC wedge, as it is not used by the majority of Cassiopei users. However, if the BASIC wedge is (for a good reason) required for an unsupported model and you are willing to test it, you can send me a request via email jd1541@hotmail.com

2.5.9 Speech synthesizer

The Cassiopei has a simple speech synthesizer that requires two additional support files to be installed onto the Cassiopei. These files are: a system file (.SYS) and a vocabulary file (.VOC). These files can be found in the download section of the Cassiopei's website.

A .SYS file is a file holding the individual samples required to speak a word, these small bit's of sound are called phonemes. Every language has it's own set of phonemes. And with these phonemes you can create any word. By playing back the proper sequence of phonemes a word is pronounced. The correct sequence of phonemes can be determined using a set of grammatical rules, but is not always accurate. The other approach is to define every word in a vocabulary. This is more accurate and also easier for the programmer who writes the speech synthesizer software (who is also the author of this document, yep, me). Unfortunately it limits the freedom of speech to the definitions in the vocabulary. In practice, this does not have to be a problem. The Cassiopei uses the vocabulary principle and therefore requires a .VOC file.

The tool to edit these files is still under development.

Cassiopei

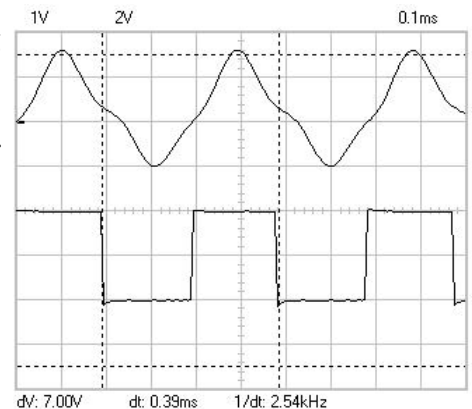
technical details

3.1 Tape protocol of the CBM's kernal loader

3.1.1 Signal encoding

Shown here is the pilot wave (a.k.a. trailing tone). It is the long beep at the start of each program. As you can see, the sinewave is not a very clean one. It looks more like a triangle. Fortunately this is not a real problem as the signal is digitized by a schmitt trigger circuit into a digital high or low value.

The upper signal is the signal as read from tape. The lower signal is the digitized value that is sent to the computer. The signal shown is the signal defined as a series of S's. The S stands for Short pulse (more about this at the bottom of this page)



The system uses three different frequencies that are always used in pairs. The only exception is the pilot tone because this is just a series of S's for several seconds. The pilot tone is used for the C64 to lock into the tape signal. This is because the tape could have been recorded on a different recorder with a different speed. Since the Pilot tone is of a defined frequency, the C64 can measure the freq. of the pilot tone and compensate for the difference in tape speed. This is why the pilot tone is heard for several seconds. Also, when a tape is started it requires time to get to the proper and stable speed. This also adds to the length of the pilot tone. You can imagine that a digital system does not have these tape speed problems and for those systems (like the Cassiopei) the pilot tone can be many times shorter, but only if the generated signals are perfectly within the range of the specifications.

Pulse duration specifications

Theoretical values for S, M and L as described in a "Data Becker" book (C64, VIC20, PET, C128)

Short : a sine of 2840Hz (\Rightarrow 352uS), this means that a high-pulse takes 176uS and that the low-pulse takes 176uS
Medium : a sine of 1953Hz (\Rightarrow 512uS), this means that a high-pulse takes 256uS and that the low-pulse takes 256uS
Long : a sine of 1488Hz (\Rightarrow 672uS), this means that a high-pulse takes 336uS and that the low-pulse takes 336uS

The C16 and Plus4 use slightly different timings

Short : a sine of 2045Hz (\Rightarrow 489uS)
Medium : a sine of 1036Hz (\Rightarrow 965uS)
Long : a sine of 520Hz (\Rightarrow 1923uS)

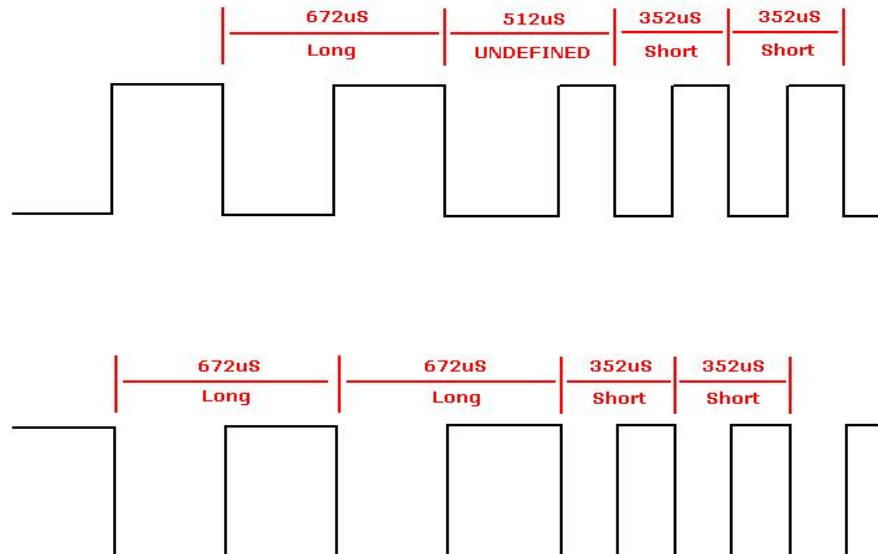
The following combinations are defined:	Pilot wave	: S
	Byte marker waves (more data follows)	: LM
	Byte marker waves (end-of-data)	: LS
	Data waves bit = 0	: SM
	Data waves bit = 1	: MS

Note: somewhere on the internet there is a rumor that the C16/Plus4 were originally designed to operate on 2MHz instead of 1MHz. And that for some reason on the very last minute in the design it was decided to lower the freq. to 1MHz. This information was derived from the fact that the tape frequencies were exactly half the freq. of the other CBM models. But if you look at the info above this isn't the really case at all. Sure 1036 looks like the halve of 1953, but if this were true then why 520Hz and not 744Hz and why 2045Hz and not 1420Hz. The real reason why these signals are different is still unknown to me. If you have more information regarding the real story of these frequencies, please contact me at jd1541@hotmail.com

3.1.2 Polarity of the signal

Now you might have wondered why is this important. Very simple. The C64 outputs a write signal to the tape that is the inverse of the signal read back from the tape. If you do not know this then this is a pitfall you can easily step into. So you can imagine that during the development of the Cassiopei I fell into this pitfall and did not understand why my signal could not be read back into the C64. The reason was simple in the end. I needed to invert it.

When you consider that **the C64 only can detect the falling edge of the read tape signal** (digitized tape output is connected to the CIA's FLAG input) it becomes clear that the C64 measures the time of a complete (digitized) sine. It does not detect the individual high or low pulse. It detects the falling edges of the pulse, which is very smart as it can be done using interrupts. Because the falling edges are triggered (and not the width of the high or low pulse) it is important that the signal is presented in the proper way. In other words in the correct polarity. When (for whatever reason) it is inverted then the C64 can no longer measure the duration of the digitized sine. Below are 2 signals. As you can see in the picture below, the first signal should be inverted. Because when it is not inverted the digitized sine is measured incorrectly and undefined timing values are detected which causes the C64 to generate a LOAD ERROR



3.1.3 How a byte is recorded

Data is transferred with odd parity (can be calculated by XOR of all Bits with starting value is 1).
Example: 1 XOR bit0 XOR bit1 XOR bit2 XOR bit3 XOR bit4 XOR bit5 XOR bit6 XOR bit7.

Pay attention to the fact that the LSB is send first. The byte marker indicates whether or not this is the last byte.

Byte marker	Bit-0	Bit-1	Bit-2	Bit-3	Bit-4	Bit-5	Bit-6	Bit-7	Odd parity
-------------	-------	-------	-------	-------	-------	-------	-------	-------	------------

The byte marker does not need to produce an end-of-data after the checksum byte of the header and repeated header because the header is a fixed length. It does produce an end-of-data marker for the data and repeated data blocks. Below is shown what happens when an end-of-data marker is transmitted (LS) there is simply nothing more send. No more bits follow. When the end-of-data byte marker is used it indicates that the block has finished.

Byte marker = LM	Bit-0	Bit-1	Bit-2	Bit-3	Bit-4	Bit-5	Bit-6	Bit-7	parity
---------------------	-------	-------	-------	-------	-------	-------	-------	-------	--------

Byte marker = LS	-	-	-	-	-	-	-	-	parity
---------------------	---	---	---	---	---	---	---	---	--------

Leader:

A 10 second leader is written on the tape before recording of the data or program commences. This leader has two functions; first it allows the tape motor to reach the correct speed, and secondly the sequence of short pulses written on the leader is used to synchronize the read routine timing to the timing on the tape. The operating system can thus produce a correction factor which allows a very wide variation in tape speed without affecting reading.

Interrecord gaps:

Interrecord gaps (2 sec.) are primarily used in ASCII files and their function is to allow the tape motor time to decelerate after being turned off and accelerate to the correct speed when turned on prior to a block read or write. Each inter-record gap is approximately two seconds long and is recorded as a sequence of short pulses in the same manner as the 10 second leader.

Trailer:

The protocol allows the use of trailers. This is just like the leader but then at the end of the file. This is optional and almost never used. The Cassiopei, does not make use of trailers as it does not add any benefits.

3.1.4 Structure on the tape

When C64 Saves data to tape it normally creates 4 tape blocks. First two are HEADER (always the same length: 202 bytes) and last two are DATA. The header and program are repeated to create the simplest form of error checking, when both versions are identical, then loading must have been successful.

LEADER	: 0x6A00 S's (approx 10 Sec) *
ID	: 0x89 0x88 0x87 0x86 0x86 0x84 0x83 0x82 0x81
HEADER	: 192 bytes (see next chapter)
CHECK	: 1 byte that represents the XOR of the data in this block

LEADER	: 79 waves (consists only of S)
ID	: 0x09 0x08 0x07 0x06 0x05 0x04 0x03 0x02 0x01
HEADER Rep.	: (repeated) 192 bytes (see next chapter)
CHECK	: 1 byte that represents the XOR of the data in this block
TRAILER	: this is not required, but doesn't hurt either

silence (roughly 0.4 seconds)

tape stops, computer waits for user to press space... within 5 seconds...

LEADER	: 0x1A00 S's (approx 2 Sec) *
ID	: 0x89 0x88 0x87 0x86 0x86 0x84 0x83 0x82 0x81
PROGRAM	: the actual program data
CHECK	: 1 byte that represents the XOR of the data in this block

LEADER	: 79 waves
ID	: 0x09 0x08 0x07 0x06 0x05 0x04 0x03 0x02 0x01
PROGRAM Rep.	: the actual program data (repeated)
CHECK	: 1 byte that represents the XOR of the data in this block

*: the Cassiopei uses 1000 pulses, this works fine. This is possible because the Cassiopei is a digital system that does not suffer from motors/tapes that require time to get up to speed or slowing down (slightly oscillating on start or finish). This is great as it greatly reduces the time required for loading our programs. Since the leader sometimes takes up more time then the actual program.

Note:

The above is for the situation that a program is saved using: **SAVE "FILENAME",1**

But when a program is saved using: **SAVE "FILENAME",1,2** then an "End-of-tape marker" is requested to be written AFTER the program that is saved. In that case the CBM computer writes an additional empty HEADER. This header will only be found if the user tries to read directly after the saved program. So actually it doesn't change the saved program in any way. It justs adds the beginning of a second empty file.

Header						
offset	Bytes used	Cassette buffer memory location				Information
		VIC20, C64	C16, Plus4	C128	PET/CBM series Cass #1	
0	1	0x033C	0x0332	0x0B00	0x027A	File type
1	1	0x033D	0x0333	0x0B01	0x027B	Load address Low
2	1	0x033E	0x0334	0x0B02	0x027C	Load address High
3	1	0x033F	0x0335	0x0B03	0x027D	End address+1 Low
4	1	0x0340	0x0336	0x0B04	0x027E	End address+1 high
5	16	0x0341 0x0350	0x0337 0x0346	0x0B05 0x0B14	0x027F 0x028E	File name
21	171	0x0351 0x03FB	0x0347 0x03F1	0x0B15 0x0BBF	0x028F 0x0339	Unused (<i>note 1</i>)

File type	
ID	Information
1	Basic file (<i>note 2</i>)
2	Data block (for sequential file)
3	Fixed address file (<i>note 3</i>)
4	Sequential file
5	End of tape maker

Note 1:

The “unused” area of the cassette buffer would normally contain all spaces (0x20), but there could also be data. For example, many turbo loaders put part of their code in here. Also the Cassiopei uses this area to load it's loader routines. The C16 and Plus 4 are using filenames of 17 characters instead of 16, this means that the 17th character is located in first byte of the unused area, but the PET's even allow 128bytes?!?!? The Cassiopei simply uses 16 chars, no fuss.

Note 2:

For the VIC20, C64, etc. files are relocatable: the start address is moved to the location pointed to by the zero page locations 0x2B and 0x2C (C64), and the end address is moved by the same amount. This happens when loading a BASIC program with **LOAD"NAME",1**

However, this can be overridden by typing: **LOAD"NAME",1,1**

In those cases, the start and end addresses provided by the header will always be used. HOWEVER for a PET computer this isn't quite the case. As files AREN'T relocatable. Files will always be loaded to the address as specified in the file.

Note 3:

Fixed address files: the start and end addresses provided by the header will always be used, no matter what the flag ,1,1 (as used in **LOAD"NAME",1,1**) is set or not. This is very useful for auto starting programs as it prevents the loading to any other address then the address specified in the header itself.

HOWEVER this functionality isn't the same on a PET, there it is best to load files with the filetype 1. If you would attempt to load a file with filetype 3 on a PET, then it simply ignores the file (took me half a day to figure that one out...)

3.1.5 Data on the tape

Now we know what signals there are on a tape and how we should interpret them you may be wondering how the actual data would look like if you would capture all the individual bytes. This paragraph gives an example of a small program and the bytes that are stored onto the tape.

```
10 PRINT"HOERA HET WERKT";  
20 GOTO 10
```

The above program results in the following data when saved to tape on a C64:

Header:

byte 000 = 01	byte 046 = 20
byte 001 = 01	byte 047 = 20
byte 002 = 08	byte 048 = 20
byte 003 = 24	byte 049 = 20
byte 004 = 08	byte 050 = 20
byte 005 = 20	byte 051 = 20
byte 006 = 20	byte 052 = 20
byte 007 = 20	byte 053 = 20
byte 008 = 20	byte 054 = 20
byte 009 = 20	byte 055 = 20
byte 010 = 20	byte 056 = 20
byte 011 = 20	byte 057 = 20
byte 012 = 20	byte 058 = 20
byte 013 = 20	byte 059 = 20
byte 014 = 20	byte 060 = 20
byte 015 = 20	byte 061 = 20
byte 016 = 20	byte 062 = 20
byte 017 = 20	byte 063 = 20
byte 018 = 20	byte 064 = 20
byte 019 = 20	byte 065 = 20
byte 020 = 20	byte 066 = 20
byte 021 = 20	byte 067 = 20
byte 022 = 20	byte 068 = 20
byte 023 = 20	byte 069 = 20
byte 024 = 20	byte 070 = 20
byte 025 = 20	byte 071 = 20
byte 026 = 20	byte 072 = 20
byte 027 = 20	byte 073 = 20
byte 028 = 20	byte 074 = 20
byte 029 = 20	byte 075 = 20
byte 030 = 20	byte 076 = 20
byte 031 = 20	byte 077 = 20
byte 032 = 20	byte 078 = 20
byte 033 = 20	byte 079 = 20
byte 034 = 20	byte 080 = 20
byte 035 = 20	byte 081 = 20
byte 036 = 20	byte 082 = 20
byte 037 = 20	byte 083 = 20
byte 038 = 20	byte 084 = 20
byte 039 = 20	byte 085 = 20
byte 040 = 20	byte 086 = 20
byte 041 = 20	byte 087 = 20
byte 042 = 20	byte 088 = 20
byte 043 = 20	byte 089 = 20
byte 044 = 20	byte 090 = 20
byte 045 = 20	byte 091 = 20
	byte 092 = 20

byte 093 = 20	byte 140 = 20
byte 094 = 20	byte 141 = 20
byte 095 = 20	byte 142 = 20
byte 096 = 20	byte 143 = 20
byte 097 = 20	byte 144 = 20
byte 098 = 20	byte 145 = 20
byte 099 = 20	byte 146 = 20
byte 100 = 20	byte 147 = 20
byte 101 = 20	byte 148 = 20
byte 102 = 20	byte 149 = 20
byte 103 = 20	byte 150 = 20
byte 104 = 20	byte 151 = 20
byte 105 = 20	byte 152 = 20
byte 106 = 20	byte 153 = 20
byte 107 = 20	byte 154 = 20
byte 108 = 20	byte 155 = 20
byte 109 = 20	byte 156 = 20
byte 110 = 20	byte 157 = 20
byte 111 = 20	byte 158 = 20
byte 112 = 20	byte 159 = 20
byte 113 = 20	byte 160 = 20
byte 114 = 20	byte 161 = 20
byte 115 = 20	byte 162 = 20
byte 116 = 20	byte 163 = 20
byte 117 = 20	byte 164 = 20
byte 118 = 20	byte 165 = 20
byte 119 = 20	byte 166 = 20
byte 120 = 20	byte 167 = 20
byte 121 = 20	byte 168 = 20
byte 122 = 20	byte 169 = 20
byte 123 = 20	byte 170 = 20
byte 124 = 20	byte 171 = 20
byte 125 = 20	byte 172 = 20
byte 126 = 20	byte 173 = 20
byte 127 = 20	byte 174 = 20
byte 128 = 20	byte 175 = 20
byte 129 = 20	byte 176 = 20
byte 130 = 20	byte 177 = 20
byte 131 = 20	byte 178 = 20
byte 132 = 20	byte 179 = 20
byte 133 = 20	byte 180 = 20
byte 134 = 20	byte 181 = 20
byte 135 = 20	byte 182 = 20
byte 136 = 20	byte 183 = 20
byte 137 = 20	byte 184 = 20
byte 138 = 20	byte 185 = 20
byte 139 = 20	byte 186 = 20

byte 187 = 20
byte 188 = 20
byte 189 = 20
byte 190 = 20
byte 191 = 20
byte 192 = 04

Data:

byte 000 = 19
byte 001 = 08
byte 002 = 0A
byte 003 = 00
byte 004 = 99
byte 005 = 22
byte 006 = 48
byte 007 = 4F
byte 008 = 45
byte 009 = 52
byte 010 = 41
byte 011 = 20
byte 012 = 48
byte 013 = 45
byte 014 = 54
byte 015 = 20
byte 016 = 57
byte 017 = 45
byte 018 = 52
byte 019 = 4B
byte 020 = 54
byte 021 = 22
byte 022 = 3B
byte 023 = 00
byte 024 = 22
byte 025 = 08
byte 026 = 14
byte 027 = 00
byte 028 = 89
byte 029 = 20
byte 030 = 31
byte 031 = 30
byte 032 = 00
byte 033 = 00
byte 034 = 00
byte 035 = 78

3.1.6 Digital flywheel technology

During the development of the Cassiopei there was no reason to implement this. Then a user of TAP-files reported a strange issue with a C64 game called "Turbo Charge". When this game was loaded, a strange bar appeared in the middle of the screen and loading did not continue. After some investigation it appeared that the game's fastloader routine was switching the motor-signal off for a very small time . Normally this has no significant impact on the speed of a real tape and loading would not be disturbed. But the Cassiopei could stop .TAP playback instantly which caused the game's loader to fail.

Now normally you would say that the game has a badly written loader. But saying so does not solve the problem. So the only way to fix it was to make the Cassiopei behave more like a real tape. In other words when the motor signal goes low, the Cassiopei no longer stops the playback, it will wait for a few extra milliseconds and if the signal is still low then stops the tape. Making it impossible to stop the digital TAP file playback instantly. The small routine that realizes this "delay" can be compared to a flywheel. When a flywheel is up to speed, it takes time to slow down. Due to it's mass it holds so much energy that it simply can't stop instantly. Hence the name "digital flywheel".

Now these kind of "problems" are very rare and I have not heard or read about another game with the same problem. But these kind of problems are interesting in their own way, first there is the troubleshooting stage. Determining what the problem is and reproducing it. Second is solving it. But third is the mystery of the origin of the problem. Why would the programmer of this tape fastloader have chosen to program it like this or was he/she even aware of this problem?

3.2 How the fastloader is loaded

As the principle of operation is for most non-PET computers is the same, this will only be about the C64. The C64 has in its memory a few locations that are not really used. The first is the unused section in the header of the standard tape protocol. This is 171 bytes that is unused but required for loading/saving. It is stored into memory locations 0x0351-0x03FB. The area 0x02A7-0x02FF (89 bytes) is unused and conveniently close to the Basic warm start vector, located at 0x0302-0x0303.

So we write a program that is stored in memory in 0x02A7-0x02FF and further in 0x0351-0x03FB.

However when we make the small part of our program 4 bytes bigger than 89 bytes, we will write into the “Print BASIC error message vector” and the “BASIC warm start vector”, we’ll fill the “Print BASIC error message vector” with the default value of 0x8B and 0xE3 and we fill the “BASIC warm start vector” with the start address of our program. Notice that 0x0302 holds the low byte of that address and that 0x0303 holds the high byte.

When the C64 has finished loading our program stored in partially free memory and partially in the cassette buffer will be called automatically, because the “BASIC warm start vector” holds the start address of our program. And then the fun begins, using the CPIO protocol the Cassiopei is requested to transfer the final program. This will load the program byte by byte until finished. The program can be executed. But first we must make sure that the following registers are set:

0x2D, 0x2F, 0x31, 0xAE must contain the low byte of the end address of our just loaded program

0x2E, 0x30, 0x32, 0xAF must contain the high byte of the end address of our just loaded program

Also memory area 0x02A7 – 0x02FF must be cleaned up. Because the C64 does not use this area and fills this area with 0x00 on reset. Some programs expect these values. And if there are other values stored then 0x00 in this area, the executed program might behave unexpected. For instance the Final Cartridge III uses 0x02AA for a flag that keep track of the basic TRACE function. And I've seen a demo flicker because of the values in this memory area. So clean using 0's and everything would be OK.

Although currently it seems to work without the suggestions below (source: Gideon Z.):

to set store the value 0x40 to zero page address 0x90 (status byte, 0x40 means end of file)

to set store the value 0x01 to zero page address 0xBA (current device number, 1=cassette)

to set store the value 0x00 to zero page address 0x35

to set store the value 0xA0 to zero page address 0x36

to set store the value 0x00 to zero page address 0x02

set stack pointer to 0xFB (LDX #\$FB followed by TXS)

Then we can finally start executing our program. And we do that by typing RUN... well not really. We poke “R”, “shift+U”, “<return>” (the short notation for RUN<return>) into the keyboard buffer. Then we set the zero page address 0xC6 to 3 because that is the number of characters we've just put into the keyboard buffer. Then we release the interrupts again (using CLI command) and we jump to the basic input loop vector (0xA480). That will notice the run command in the keyboard buffer and execution begins just as if we manually typed RUN<return>.

We use the return trick simply because this proved to be the most effective, typing run appears to set some basic flags we otherwise needed to set in our loader program at the cost of many bytes of programming space. Which is something that we simply cannot afford to do in the limited area 0x02A7-0x02FF and 0x0351-0x03FB.

The PET/CBM 2000, 3000, 4000, 8000 models work differently then the C64 and therefore they cannot use the same method of auto-starting, in order to auto-start, the Cassiopei needs to modify the stack of the PET/CBM computer, this is a very tricky exercise and is therefore not implemented. Modifying the stack requires absolute knowledge of the system the program is running on and because there are so many types and variations regarding the contents of the stack of these different models this functionality will never be developed. Sorry, you must load with shift+RUNSTOP (BASIC 2) or type RUN (BASIC 4)

3.3 CPIO protocol

The CPIO (Cassette Port Input Output) protocol is a protocol designed to interface a slave device to a CBM computer (it started with a C64) using the cassette port. The intention of the CPIO is to have a fast synchronous and bi directional (half duplex) protocol to transfer data between the CBM computer and it's slave.

CPIO is defined as a synchronous protocol to make sure that the VIC (the C64's video processor) can continue to operate without causing problems of disturbing the timing of the data transfer protocol. In other words during the fast data transfer of the Cassiopei there is no need to disable the screen (to prevent the VIC from interrupting the timing). Disabling of the screen is known to be used for LOADING and SAVING from the datasette during normal use. The CPIO protocol does not require disabling the screen. So You can have a high data transfer rate without errors and with all the fancy things on your screen visible.

The slave device can be all sort of devices. Using the CPIO in combination with a fast micro-controller connected to the cassette port of a CBM computer, the number of possible applications becomes unlimited. For example an AD-converter, simple digital IO, fast serial port UART, counter, etc. But also file loading becomes possible. Important to know is that the CPIO device must be capable of transferring data over the cassette port as if the device is a datasette. This makes it possible to have the application as well as the software for that application into one device. Because the CPIO is a fast protocol, the application software can be loaded according a 2 step procedure. The user simply types LOAD and the CBM computer initiates a load sequence as if it would load from a real datasette. The device loads a small piece of software into the CBM which incorporates the CPIO protocol. Because this code will auto start the user does not have to do anything anymore until the application becomes fully active.

The CBM computer is the slowest device and therefore determines the max. speed of the CPIO protocol.

3.3.1 The CPIO protocol's signals (synchronous 8-bit)

Master : CBM computer

Slave : Cassiopei or any other device using the CPIO protocol

Attention:

This signal indicates to the slave the desired state of communication. When this signal goes high, the slave is expected to respond with READY = low as soon as possible. When this signal goes low, the slave must finish it's current byte read/write action and then stop communication.

This signal is actually the motor power line. It is driven by a power transistor, which is relatively slow. Also this line should be loaded with a few hundred ohm in order to make the high to low transition faster (820 Ohm is sufficient, this will dissipate approx. 1/8 Watt when the ATTENTION is high, it is not much compared to the current of the motor, but it is enough to drain the voltage quickly). Because this signal is so slow, we must clear this signal before we send or read the last byte. Deactivating attention takes approx 150uSec on a C64. By deactivating the attention signal just before the last byte is send/received the attention signal will become deactivated in the middle of the byte (because it is so slow). What means that it is stable when the byte has finished. Which is exactly the moment that the attention signal is being sampled for it's state by the connected slave device. This is the best way of working around this slow signal without compromising the data transfer speed.

low = no communication

high = communication with slave

default = low

Clock:

This signal makes the data transfer synchronous required for high speed communication. The clock line is controlled by the master. CLOCK will go low as soon as the slave responds with a READY = low.

low = data setup (slave/master are expected to write directly after the falling edge)

high = data stable (slave/master are expected to read directly after the rising edge)

default = high

Data:

bi-directional data line. This line is Read or Write, the master determines the direction of communication. This is defined in the first byte (mode byte) of each communication session.

low = logical '0'

high = logical '1'

default = high

Ready (active low):

This signal indicates if the slave is ready to communicate, as long as this signal is high, the master will not start with lowering the CLOCK line. Please note that the C64 can only detect the falling edge of this signal (READY is connected to the Flag input of the CIA, which is an edge negative triggered input pin).

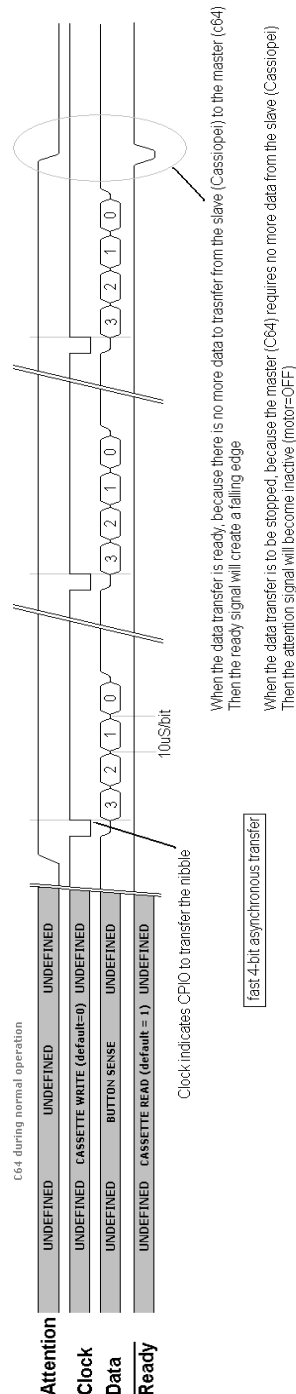
low = slave is ready for communication

high = slave is busy

default = high

3.3.2 The CPIO protocol's signals (asynchronous 4-bit)

The 8-bit format as described on the previous pages is very reliable as it is a synchronous protocol. The clock keeps the data transfer perfectly aligned between master and slave. So that protocol is very stable and safe, no matter what disturbs the timing, no data will ever get lost. Unfortunately, generating a clock signal and its detection takes time, so this protocol isn't optimal when it comes to speed. For some situations where speed IS of the essence it might be useful to have an alternative. That situation is sample playback. CBM computers that feature a sound chip can playback digitized sound, a.k.a. Digis. These sound samples are 4 bits, due to the limits of the volume register of those sound chips. So a fast protocol of 4 bits is described below. Be aware that this protocol requires very accurate timing routines for the slave and the master. Currently this is only used for the transfer of 4-bit sound.



3.4 CPIO messages definitions

The Cassiopei is a device that has many functions. Therefore the master (the Commodore computer also referred to as CBM) needs to tell the Cassiopei in which mode it should operate. Because it can do so many, the CBM (or actually the software of the CBM) must be aware of the exact flow of data.

Therefore the CPIO protocol demands that the first byte in each session is the mode-byte. This byte describes the slaves mode of operation. All bytes that follow can be read or write.

3.4.1 Command: Load

This function is intended for the CPIO menu program (sometimes referred to as configuration program) and it not recommended for other purposes.

Load file CPIO_LOAD read PROGRAM from slave, this mode is useful as it is faster then the standard tape protocol										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	0	0	0	0	0	0	0	0
2	Slave write	data	Low byte of file's last byte address in C64 RAM							
3	Slave write	data	High byte of file's last byte address in C64 RAM							
4	Slave write	data	Low byte of LOAD address of file's first byte's address in C64 RAM							
5	Slave write	data	High byte of LOAD address of file's first byte's address in C64 RAM							
6-...	Slave write	data	The file to be loaded							
End	Slave write	data	This byte may be discarded, it is just a dummy in order to keep our 6502 loops simple, since this last byte is a dummy, we do not need to process it and the loading loop does not need to check for it.							

For the VIC-20, it is sometimes required to load to a different address other then specified in the file. In this case the function below make that partially possible. It still requires the software in the VIC-20 (menu program) to have a decicated loader that overrules the file's load address.

Load file (but forces the filetype to PRG) CPIO_FILETYPE_TO_PRG_THEN_LOAD read PROGRAM from slave, this mode is useful as it is faster then the standard tape protocol										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	0	0	0	0	0	0	0	0
2	Slave write	data	Low byte of file's last byte address in C64 RAM							
3	Slave write	data	High byte of file's last byte address in C64 RAM							
4	Slave write	data	Low byte of LOAD address of file's first byte's address in C64 RAM							
5	Slave write	data	High byte of LOAD address of file's first byte's address in C64 RAM							
6-...	Slave write	data	The file to be loaded							
End	Slave write	data	This byte may be discarded, it is just a dummy in order to keep our 6502 loops simple, since this last byte is a dummy, we do not need to process it and the loading loop does not need to check for it.							

3.4.2 Command: Data load

This routine is for loading data from a file, for instance if you program a game that requires data which is stored in a file (a screen for instance, or a (set of) images or perhaps a playfield). This command is not intended to load a game or program into the CBM memory. Although not impossible, the CPIO_LOAD command is designed for that function. This command loads only from data files.

Load from data file										
Before we can load data, we must specify the filetype, offset and filename using CPIO_PARAMETER										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	0	0	0	0	1	1	0	1
2	Master write	data	MSB of 24bit offset value							
3	Master write	data	... of 24bit offset value							
4	Master write	data	LSB of 24 bit offset value							
5	Master write	data	1st ASCII character of the filename							
6	Master write	data	2nd ASCII character of the filename							
7	Master write	data	3rd ASCII character of the filename							
..							
n	Master write	data	n-th ASCII character of the filename							

Load from data file										
CPIO_DATALOAD										
Read data from a specified file stored on the Cassiopei's flash memory										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	0	0	0	0	0	0	0	1
2	Slave write	data	0 = Error: file could not been found 1 = Ready to transfer data							
3	Slave Write	data	First byte read from the specified position in the file							
..							
n	Slave write	data	n-th byte read from the specified position + n in the file							

How to load data from a file using the cassiopei BASIC wedge (C64):

This functionality is not available.

3.4.3 Command: Data save

Before data can be written to the Cassiopei's flash file using CPIO_DATASAVE the command CPIO_CREATEFILE needs to be called first. This command sets up the header information of the file and makes sure that all pointers are pointing to the correct file locations.

Create a file CPIO_CREATEFILE Create a file to which data can be saved (using CPIO_DATASAVE)										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	1	0	0	0	0	0	0	0
2	Master write	data	Filename 1 st character							
3	Master write	data	Filename 2 nd character							
/	/	/								
33	Master write	data	Filename 32 nd character							
34	Master write	data	MSB of filesize							
35	Master write	data	XSB of filesize							
36	Master write	data	LSB of filesize							

The filename can be up to 32 characters long. All characters need to be defined, therefore if shorter filenames are desired, then the remaining length must be padded with spaces until all 32 characters are defined.

The 3 last bytes of indicate the exact filesize (meaning the size of all the data stored inside the file). It is very important that this value is correct otherwise the Cassiopei manager will see it as a corrupted file. Therefore the CPIO_DATASAVE command can only be used in situations where the filesize is exactly known. In other words, it is not suited for writing logfiles or other types of files that have an unknown size at the moment that the file is created.

The CPIO_DATASAVE command is only to be called AFTER CPIO_CREATEFILE has been called first. The minimum payload of this command is 1 byte, the max payload is 128 bytes.

Save data to file CPIO_DATASAVE Save data to an opened file (created using CPIO_CREATEFILE)										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	1	0	0	0	0	0	0	1
2	Master write	data	1 st Data byte saved to file							
3	Master write	data	2 nd Data byte saved to file							
/	/	/								
129	Master write	data	128 th Data byte saved to file							

How to save data to a file using the cassiopei BASIC wedge (C64):

This functionality is not available.

3.4.4 Command: File info from flash

This function is intended for the CPIO menu program (sometimes referred to as configuration program) and it not recommended for other purposes.

Load File info from flash										
CPIO_DIRECTORY_FLASH										
request the file info of file with index ..., useful for a file selection menu and showing directories (only user files of the type .PRG are supported)										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	0	0	0	0	0	1	0	0
2	Master write	File type								
3	Master write	index	Value 0-255							
4	Slave write	Status	0= file not found 1=file found							
5	Slave write	data	1 st char of filename (if not used then value 0x20=space)							
6	Slave write	data	2 nd char of filename (if not used then value 0x20=space)							
The returned data is identical to the file structure of .PRG files										

3.4.5 Command: Get remaining space

This function is intended for the CPIO menu program (sometimes referred to as configuration program) and it not recommended for other purposes. This requests the free space on the flash memory in blocks.

Get remaining space CPIO_GETREMAININGSPACE request the file info of file with index ..., useful for a file selection menu and showing directories (only user files of the type .PRG are supported)										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	0	0	0	0	0	1	1	0
2	Slave write	data	Low byte of the available space in blocks							
3	Slave write	data	High byte of the available space in blocks							

3.4.6 Command: Simulate button

This function is intended for the CPIO menu program (sometimes referred to as configuration program) and it not recommended for other purposes.

Simulate button										
CPIO_BUTTON										
read PROGRAM from slave, this mode is useful as it is faster then the standard tape protocol										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	0	0	0	0	0	1	1	1
2	Master write	function	Indicate what button function needs to be simulated							

Function byte		
Value	Simulated button	Information
0x01	PLAY	This value simulates that the PLAY button is pressed
0x11	PLAY-C64	<p>This value simulates that the PLAY button is pressed, BUT the configuration of the computer model is changed to C64. This change is not saved to EEPROM and remains in RAM until the Cassiopei is reset.</p> <p>The settings PAL or NTSC will be derived from the currently used computer model. So if the Cassiopei is configured for an NTSC C128, then the used C64 mode will also be NTSC.</p> <p>When the simulated play button has fulfilled its task, the Cassiopei will switch back to its original computer model setting.</p>
0x02	MENU	This value simulates that the MENU button is pressed
0x12	MENU-C64	This value simulates that the MENU button is pressed, see also PLAY-C64

3.4.7 Command: Read ADC

Read ADC value										
CPIO_ADC										
read from ADC mode, this device will respond with the results of the ADC of channel 1										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	0	0	0	0	1	0	0	1
2	Master write	channel	Value that indicates the ADC channel to be read							
3	Slave write	data	2 MSB's of the 10 bits ADC result (the first six bits are 0)							
4	Slave write	data	8 LSB's of the 10 bits ADC result							

The possible ADC channels are:

0 : connected to pin 3 of the expansion connector

1 : connected to pin 5 of the expansion connector

See the section Expansion port for more detail on how to use these inputs correctly.

How to get and ADC value using the cassiopei BASIC wedge (C64):

Load the cassiopei BASIC wedge into you CBM computer and start it.

Using the command `A=ADC,0:PRINT A` you can do the AD conversion and print it to the screen.

But this is not very useful. We want to work with the ADC value, because your application most likely requires some scaling of the value to convert it into a temperature or voltage. The example below uses $R1=10K$ ohm and $R2=1K$ ohm.

Example:

```
5 SYS 49500 :REM ACTIVATE WEDGE (C64)
10 REM READ THE VOLTAGE VALUE OF THE ADC USING VOLTAGE DEVIDER
20 MAX=1023:REM MAX VALUE OF ADC
30 VCC=3.30:REM CASSIOPEI WORKING VOLTAGE
40 BIT=VCC/MAX:REM THE VALUE OF A SINGLE LSB
50 SCALE=(1000+10000)/1000
60 REM -----
100 A=!ADC,0 :REM get the value from ADC channel 0
110 U = A * BIT * SCALE
120 PRINT"VOLTAGE =";LEFT$(STR$(U),5)
130 GOTO 100
```

Note:

Line 30: improve accuracy using the measured voltage using a voltmeter on pin 7 of the exp. connector

Line 50: improve accuracy check the exact values of the used resistors

Line 120: the LEFT\$ function is used to limit the number of printed digits, experiment with value 5

3.4.8 Command: Read / Write EEPROM

The EEPROM function is not really reading/saving the data from/to EEPROM as the 18F26J50 micro-controller used in the Cassiopei has no available EEPROM. But the Cassiopei has several Mbytes of flash memory, so storage is not really a problem. But one thing that must be kept in mind is that Flash can only be written in blocks. While EEPROM can be written in bytes. In order to change a value in a flash block the entire block need to be read, change the desired byte and then rewritten back to flash. Because the memory of the Cassiopei micro-controller is smaller then the size of a flash block it is impossible to read a single block into memory in order to modify it. Although it is possible to work around this it is quite inefficient considering the wear on the flash block holding the data.

This functionality is therefore reserved for storing the Cassiopei's settings. This function will not be available in the BASIC wedge. If storage room is required it is best to connect a I2C EEPROM memory, these are small 8 pin IC's that can hold up to many Kilo Bytes of data. The cassiopei's expansion port has a I2C databus and the Cassiopei's BASIC wedge has I2C commands to easily use this bus.

To reduce wear on the flash memory a crude method of wear leveling has been implemented but at the cost of the usable memory locations. This is no problem as the Cassiopei only requires a few registers for storing it's settings. This results in a system that is capable of writing a single value every 5 minutes for 16 years before the used flash memory cells are used beyond their guaranteed specifications. In practice this simply means that the simulated EEPROM never wears out.

Location 0x00 is used for storing the Cassiopei operating mode

Location 0x01 is used for storing the Cassiopei file index

Location 0x02 is used for storing the computer model that is used with the Cassiopei

Location 0x03 is used for storing the feedback volume setting

Location 0x04 is used for storing the C128's mode of operation (C64 mode or C128 mode)

Read EEPROM CPIO_EEPROM_RD read from byte from EEPROM										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	0	0	0	0	1	1	1	1
2	Master write	address	X	X	X	X	Address (15 locations)			
3	Slave write	data	Data read from EEPROM							

Write EEPROM CPIO_EEPROM_WR write data to EEPROM										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	1	0	0	0	1	0	1	0
2	Master write	address	X	X	X	X	Address (15 locations)			
3	Master write	data	Data that needs to be written to EEPROM							

3.4.9 Command: DTMF generator

Generate DTMF signal CPIO_DTMF										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	1	0	0	0	1	0	0	1
2	Master write	data	X	X	X	X	4 bit value of DTMF tone			

The DTMF signal is outputted on pin 2 of the cassiopei's expansion port, see the section Expansion port for more detail on how to use this correctly.

DTMF signals are the “musical notes” you hear when you press the number keys on your telephone. DTMF stands for Dual Tone Multiple Frequency, it is used by telephone companies to determine the telephone number you are dialing. This sound of it is very distinctive and recognized by almost everyone who has ever used a telephone. There are 16 possible notes 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

How to generate DTMF tones using the cassiopei BASIC wedge (C64):

Load the cassiopei BASIC wedge into you CBM computer and start it.
Using the command !DTMF,KEY command you can generate DTMF signals

Example:

!DTMF,0<RETURN> This generates the “tone”you would hear when you dial 0 on your telephone
!DTMF,3<RETURN> This generates the “tone”you would hear when you dial 3 on your telephone

But if you want to generate a sequence of tones a program would be better, example shown below:

```
5 SYS 49500 :REM ACTIVATE WEDGE (C64)
10 RESTORE
20 READ A
30 IF A=255 THEN 100
40 !DTMF,A
50 FOR L = 0 TO 250 : NEXT L
60 GOTO 20
100 END
200 DATA 0,6,1,2,3,4,5,6,7,8
299 DATA 255
```

Note:

line 30: determines if the end of sequence value is found, as 255 is not a valid key.

line 40: plays the tone

line 200: holds the sequence of notes

line 299: end of sequence marker

Regarding syntax of the DTMF values A,B,C,etc use the value 10,11,12,etc

Speech synthesizer in order to synthesize speech, there are some parameters required, this uses the command: CPIO_PREPARE										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	0	0	0	0	1	1	0	1
2	Master write	data	Destination (0=CBM, 1=PWM, 2=CBM and PWM)							
3	Master write	data	First ASCII character of string to be spoken (case is unimportant)							
4	Master write	data	Second ASCII character of string to be spoken (case is unimportant)							
5-end	Master write	data	... ASCII character of string to be spoken (case is unimportant)							

If the destination value of 0 is chosen the speech is outputted through the CBM's audio device in the form of a 4-bit digitized sample. This requires no additional cables or wires.

If the destination value of 1 is chosen then the speech signal is outputted on pin 2 of the cassiopei's expansion port, see the section Expansion port for more detail on how to use this correctly.

If destination value is 2, the both output methods are used.

Now that the parameters for the speech synthesizer are transferred to the Cassiopei, it is time to playback the speech. This (may) require the 4bit playback routines depending on the chosen destination. This command responds with a byte to indicate that it is ready to playback the speech. Or not in case the file could not been found. It is important that there are no other CPIO command(s) given between CPIO_PARAMETER and CPIO_SPEECH.

Speech synthesizer CPIO_SPEECH										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	1	0	0	0	1	1	1	0
2	Slave write	data	0 = Error: sample file could not been found 1 = Ready to play sample							

How to generate speech using the Cassiopei BASIC wedge (C64):

Load the cassiopei BASIC wedge into you CBM computer and start it.

Using the command !SAY,destination,"text to be spoken" you can generate speech

Make sure that you've installed the systemfile (.SYS, this is a file holding the phonemes for the actual speech) and the vocabulary file (.VOC, this is a file that describes how the words are to be pronounced). Without these files speech is not possible. These files can be downloaded from the download section of the Cassiopei's website. For more information regarding these files and the related tool see the earlier section "speech synthesizer" in this manual.

Example:

```
!SAY,0,"GREETINGS PROFESSOR FALKEN, SHALL WE PLAY A GAME"<RETURN>
```

But string variables can also be used. Normal variables cannot be spoken unless converted to a string variable, which is not difficult as you will see further on. This is done because formatting of the spoken value is almost always required. A variable that holds the result of a calculation is most likely to be a float. And that value can be very long and annoying when spoken. For instance *"the current temperature is 21.342145343556 degrees Celsius"*. Where normal people would prefer *"the current temperature is 21.3 degrees Celsius"*. So the temperature value needs to be formatted in order to speak it in a sensible manner. The example below shows how to do that.

```
5 SYS 49500 :REM ACTIVATE WEDGE (C64)
10 T=22.3421234
20 T$=LEFT$(STR$(T),5)
30 !SAY,0,"THE CURRENT TEMPERATURE IS"
40 !SAY,0,T$:!SAY"DEGREES CELCIUS"
```

Note:

The string variable in this DOS wedge is limited from A\$ to Z\$, this gives you 26 possible variables to use with the !say command. Which is more than enough. In practice, you most likely will never need more than 1. As the variable can be recycled over and over again. If you enter a variable with a different name for instance TEXT\$ then you'd get an error, so stick to the convention of a single letter followed by the \$-sign for any string variable used with the !SAY command. This is done for simplicity of the wedge.

Note:

You can only say the words that are defined in the vocabulary. If you enter a word that is not defined the speech generator inside the cassiopei will indicate this by sounding a "beep" instead of the requested word.

Note:

Computers that do not feature a decent audio chip cannot use the mode 0 and 2. And therefore the only method of audio playback/speech is through the PWM output of the Cassiopei.

3.4.11 Command: Sample player

Sample player in order to play a sample, there are some parameters required, this uses the command: CPIO_PREPARE										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	0	0	0	0	1	1	0	1
2	Master write	data	MSB of 16bit offset value (value*256 samples to skip before playing)							
3	Master write	data	LSB of 16bit offset value							
4	Master write	data	MSB of 16 bit end value (value*256 samples to play, 0=play full sample)							
5	Master write	data	LSB of 16 bit end value							
6	Master Write	data	Destination: 0=CBM (requires CBM audio device and sample playing routines) 1=PWM (requires connection to expansion port of Cassiopei) 2=PWM and CBM							
7	Master write	data	1st ASCII character of the filename of the WAV file							
8	Master write	data	2nd ASCII character of the filename of the WAV file							
9	Master write	data	3rd ASCII character of the filename of the WAV file							
..							
n	Master write	data	n-th ASCII character of the filename of the WAV file							

The offset and end value of the sample are 16-bits values when passed to the Cassiopei, this is done to make it easier for the CBM computer to handle the values. As the CBM can't handle 24bit values without making the wedge considerably larger. However there is no real need for a 24 bits value. Because the 16bit value is multiplied by 256, meaning that you can address a sample of a size of 16MByte, which is quite large. The resolution of the offset and end address is 0.032sec (256 samples at 8KHz = 0.032sec), which is not noticeable.

The filename must be at least one character long, and max 32. However the full filename is not really required. As the system will look for the first file that matches the given string. If the filename is longer then the given string but does match, then that file will be used. The end of the string is normally indicated by the value 0. However this value may be left out as the Cassiopei detects the last character in the communication and therefore it knows that the string has ended also. So if the value 0 has not been send, the Cassiopei adds the value 0 to the end of the string.

The sample data is output on pin 2 of the Cassiopei's expansion port, see the section Expansion port for more detail on how to use this correctly. Also the sample data is send to the CBM computer using a 4bit asynchronous protocol, so that it is possible to play the sound through the CBM's sound device. This make it possible to listen to the audio without connecting anything to your cassiopei's expansion port.

Now the settings for sample playback are send to the Cassiopei, it is time to playback the sample. This may require the 4bit playback routines depending on the chosen destination. This command responds with a byte to indicate that it is ready to playback the sample. Or not in case the file could not been found. It is important that there are no other CPIO command(s) given between CPIO_PARAMETER and CPIO_PLAYSAMPLE.

Sample player CPIO_PLAYSAMPLE										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	0	0	0	0	1	1	1	0
2	Slave write	data	0 = Error: sample file could not been found 1 = Ready to play sample							

How to play samples using the cassiopei BASIC wedge:

Load the Cassiopei BASIC wedge into you CBM computer and start it.

Using the command !SAMPL,<start>,<stop>,<destination>,"<filename>" you can play back the WAV. You must have loaded a sample file onto the Cassiopei's flash memory, the size is not important as long as it fits onto the remaining space of the flash memory. The <start> and <stop> values are the byte position/256 in the sample file. The sample must be 8bit, 8KHz MONO. It will be played back onto the expansion port in 8 bits PWM and it is send to the CBM in a 4 bit format. This is done because most CBM computers cannot produce more then 4bits digital sound (without making programming too difficult).

The main advantage of the PWM output is it's better quality of sound and it does not require any effort of the CBM computer, so after the command of playing audio to the PWM output, the CBM can continue with (non Cassiopei related) tasks.

Example:

!SAMPL,0,0,0,"S01"

play a sample from start to finish to the CBM audio device, the filename starts with "S01"

!SAMPL,0,0,1,"S01"

play from start to finish to the PWM output of the Cassiopei, the filename starts with "S01"

!SAMPL,0,0,2,"S01"

play from start to finish to the PWM output and the CBM audio device, the filename starts with "S01"

!SAMPL,200,100,0,"S01"

skip first 200*256 samples, stop playing after 100*256 samples, output is CBM audio device, the filename starts with "S01"

Note: because the Cassiopei does not know the exact filesize it is very important that the offset value is not set to a point further then the length of the sample. Because if the filepointer is set to a location further then the data, there is no useful data to be read. Since you may assume that a programmer knows the size of it's own samples, this is not really a problem. If the chosen offset is too large there will be no

sound (the system appears to halt), run/stop + restore to stop the playback and try again with a smaller offset value. Pressing the Cassiope is reset button also stops the sample playback.

Generate SineWave CPIO_SINEWAVE										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	1	0	0	0	1	0	0	0
2	Master write	data	High byte of integer value of frequency of sine wave (Hz)							
3	Master write	data	Low byte of integer value of frequency of sine wave							
4	Master write	data	High byte of integer value of duration of sine wave (msec.)							
5	Master write	data	Low byte of integer value of duration of sine wave							

The sinewave signal is outputted on pin 2 of the cassiopei's expansion port, see the section Expansion port for more detail on how to use this correctly.

The frequency range of this function is between 1Hz and 5KHz. Although higher frequencies can be entered in the function and generated by the generator, distortion of the generated sinewave will become noticeable. High-end users of low frequencies must keep in mind the generated sine wave is build from a table of 256 sinewave samples. Therefore additional filtering may be required.

How to generate sinewave using the cassiopei BASIC wedge (C64):

Load the cassiopei BASIC wedge into you CBM computer and start it.

Using the command !SINE,1000,2000 you can generate a 1KHz sinewave that lasts for 2000 milli seconds (=2seconds).

Generate KlikAanKlikUit 13Bits signal CPIO_KAKU13										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	1	0	0	0	1	0	1	1
2	Master write	data	House (0=A, 1=B, 2=C, 3=D) Slide switch on the back of the remote control							
3	Master write	data	Group (1=I, 2=II, 3=III, IV=4) slide switch on the front of the remote control When this switch is not on the receiver use value 1 (=I)							
4	Master write	data	Button (1,2,3,4) the button on the remote control							
5	Master write	data	State (0=OFF, 1= ON)							

This command sends signals just like the YCT-102 (left) and PA3-1000R (right) KlikAanKlikUit remote control. The picture right shows a starterset which includes the PA3-1000R as you can see this remote does not offer the Group setting slide switch (which is no problem at all).

Using the Cassiopei's BASIC wedge you can very easily write your own time schedules on your C64 computer. Home automation has never been so easy/fun.



How to send a signal using the cassiopei BASIC wedge (C64):

Load the cassiopei BASIC wedge into you CBM computer and start it.

Using the command !KAKU you can send the KlikAanKlikUit signal.

The required parameters are !KAKU,<House>,<Group>,<Button>,<State>

House : 0=A, 1=B, 2=C, 3=D, etc. (the house code A,B,C,D)

Group : 1=I, 2=II, 3=III, 4=IV (considering most receiver don't offer this settings, choose 1)

Button : 1=1, 2=3, 3=3, 4=4, etc. (the actual button you would press on the remote control)

State is: 0= OFF, 1=ON (ON or OFF, and that's it, dimmers cannot be used)

Example:

To switch ON the light of a first generation KlikAanKlikUit receiver with ID: D2 type the command:

!KLI13,3,1,2,1 <RETURN>

To switch OFF the light of a first generation KlikAanKlikUit receiver with ID: D2 type the command:

!KLI13,3,1,2,0 <RETURN>

Note:

When more then 1 KlikAanKlikUit command is send, for instance when controlling multiple lights, then make sure that there is enough time between these signals. When the signals aer send to close to eachother then the receiver modules might not respond properly. This is simply the way that these devices work and the Cassiopei cannot alter this. A simple method of delaying is sufficient for example:

100 FOR D=0 TO 50:NEXT D

You are free to experiment with the value 50, a greater value makes the delay longer and vice versa. The delay in this example is very small and a longer delay is not really required. Depending on your programs implementation the delay might not even be required, simply because the Commodore BASIC language isn't that fast.

Generate KlikAanKlikUit 32Bits signal CPIO_KAKU32										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	1	0	0	0	1	0	1	1
2	Master write	data	House (0-255) Choose a value that you like and that is not in use by your neighbors. From this value a 26 bit house address value will be created, it allows you to have 255 other C64 users with the same system without interfering with each other. Cool, don't you think?							
3	Master write	data	Button (1,2,3,4) the button on the remote control							
4	Master write	data	State (0=OFF, 1= ON) Dimmer mode (128= min. light level, ..., 143=max light level)							

This command sends signals that are compatible with the AYCT-102 and ACD-300 KlikAanKlikUit series of devices. The picture below is the dimmer set consisting of the AYCT-102 remote control and the ACD-300. The ACD-300 is a dimmer, but as you can see, dimming is not that easy as you have only 2 buttons ON and OFF. So in order to dim your lights to a specific value you'd need a different remote control, one that offers a slider of some kind, using your ipad or mobile phone that has a special app bladeiebla. This remote only let's you switch the light ON, press ON again and it will dim until you press ON again and then it will stick to that dim value. When you switch it OFF it remembers the last dim value. So it works, but it could be much better. And it will be better because your C64 can dim directly to the proper value, using the Cassiopei and modified remote and a simple BASIC program in combination with the Cassiopei's BASIC wedge.



How to send a signal using the cassiopei BASIC wedge (C64):

Load the cassiopei BASIC wedge into your CBM computer and start it.

Using the command !KAKU you can send the KlikAanKlikUit signal.

The required parameters are !KAKU,<House>,<Group>,<Button>,<State>

House : 0 – 255 (this will be your house code that you'll receivers must learn, not critical as long as you use the same value every time)

Button : 1=1, 2=3, 3=3, 4=4, etc. (the actual button you would press on the remote control)

State is: 0= OFF, 1=ON (On or OFF, or dimmed if value is 128 - 143)

Example:

To switch ON the light of a second generation KlikAanKlikUit receiver type the command:

!KLI32,187,0,1 <RETURN>

To switch OFF the light of a second generation KlikAanKlikUit receiver type the command:

!KLI32,187,0,0 <RETURN>

To dimm a light to a specific value type the command:

!KLI32,187,0,128 <RETURN> (128=lowest light level, 143=max light level)

Because there are only 16 dimming steps, the dim resolution isn't perfect, but for most purposes it will be sufficient. This is a limitation of the KlikAanKlikUit protocol

However, chances are that this doesn't work yet. Because the receiver must have “learned” to accept the chosen house and button value. Although there is a chance in 1:67 million that it has. So keep on reading and you'll know how to solve this.

In this example below we will use house value 187, but this may be any value that you'd prefer, the house value is designed to be different for every household. Which makes sense if you imagine if you and your neighbours are using the same system. But the house code is originally a 26bits value, here it is only eight bits. So the cassiopei changes that 8 bit value into a 26bits value. It does not really matter what the house code is, as long as it is not the same as other users withing your systems receiving range. And it does not really make sense to let you enter such a large number of 26 bits if the system has a range of max. 150 meters, so a simple value for much easier use.

Once you've chosen your house code, the receiver must learn it. This can be done very easily, press the button on your receiver (using a pen) and send the code as shown below. This will switch the light ON and the receiver will recognize this and remember it. For more details read the manual of your KlikAanKlikUit receiver. For every light you want to switch independently you must choose a different button value. If you want to control more then 16 lights, simply change the house code and you've got another 16 lights to control. Theoretically you could independently control $256 \times 16 = 4096$ lights (or other devices you could connect to your mains power supply). That's a lot, more then you'll ever need (or can afford).

Note:

When more then 1 KlikAanKlikUit command is send, for instance when controlling multiple lights, then make sure that there is enough time between these signals. When the signals aer send to close to eachother then the receiver modules might not respond properly. This is simply the way that these devices work and the Cassiopei cannot alter this. A simple method of delaying is sufficient for example:

```
100 FOR D=0 TO 50:NEXT D
```

You are free to experiment with the value 50, a greater value makes the delay longer and vice versa. The delay in this example is very small and a longer delay is not really required. Depending on your programs implementation the delay might not even be required, simply because the Commodore BASIC language isn't that fast.

3.4.15 Command: I2C

I2C:

I2C is a protocol used by many devices, therefore it is perfect for expansion of your hardware. Using I2C it is easy to add more simple IO's or complex circuitry that performs difficult tasks. The most important thing is that I2C is a bus protocol. This means that you can connect multiple devices onto the same bus. As long as the wiring is less then 1meter and the devices all have unique addresses.

The I2C signals are connected to pin 4 and 6 of the cassiopei's expansion port, see the section Expansion port for more detail on how to use this correctly.

I2C communication CPIO_I2C										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	1	0	0	1	0	0	0	0
2	Master write	function	I2C function of the cassiopei (what we want to do on the I2C bus)							
3	Master W/R	data	Data depends on I2C function (this byte is not always required)							

Function and Data byte		
I2C function	Byte	Information
CPIO_I2C_STOP	2	0x00 = Send stop condition to I2C slave
	3	No data byte required
CPIO_I2C_ADR_W	2	0x01 = write I2C address to slave and indicate sending data TO the I2C slave
	3	Data byte holds the slave address
CPIO_I2C_ADR_R	2	0x02 = write I2C address to slave and indicate reading data FROM the I2C slave
	3	Data byte holds the slave address
CPIO_I2C_PUT	2	0x10 = write I2C data to slave
	3	Data byte holds the data to be send to the I2C slave
CPIO_I2C_GET	2	0x20 = read I2C data from slave and acknowledge
	3	Data byte holds the data that is read from the I2C slave
CPIO_I2C_GETLAST	2	0x21 = read I2C data from slave, no acknowledge indicates this byte as last byte
	3	Data byte holds the data that is read from the I2C slave

How to communicate with an I2C device using the Cassiopei's BASIC wedge (C64):

The Cassiopei's BASIC wedge allows communicating with a I2C device using 6 simple BASIC commands. Using the I2C bus requires full knowledge of the device that is being communicated with. The exact protocol the device required is specified in the data sheet of the I2C device.

`! I2CAW , <adr>`

This function is called I2C open Address Write.

Opens the I2C device, that has the specified I2C address, for writing to this device.

`! I2CAR , <adr>`

This function is called I2C open Address Read.

Opens the I2C device, that has the specified I2C address, for reading from this device.

`! I2CPU , <data>`

This function is called I2C PUt data.

Sends data to the opened I2C device.

`DATA= ! I2CGE`

This function is called I2C GEt data.

Reads data from the opened I2C device and indicates that more data will be read.

`DATA= ! I2CGL`

This function is called I2C Get Last data.

Reads data from the opened I2C device and indicates that this was the last byte read from this device.

`! I2CST`

This function is called I2C STop.

Stop communicating with the I2C device

Example (C64):

Controlling a stepper motor in half step mode using the circuit as shown below (PCF8574A, ULN8003 and a stepper motor from an old printer) is not difficult when you use the basic wedge. Let's assume that the stepper motor is connected to the 4 lowest bits of the IO-expander, bit 0,1,2 and 3.

The only problem with stepper motors is that they need a continuous stream of signals in order to keep stepping. So controlling the stepper motor might lock up your program for the same time the motor must be turning

Keep in mind that BASIC is not the fastest language, the program below takes 7.8mSec per step, meaning that a stepper motor cannot rotate very fast. So think twice before you choose a stepper motor.

```
5 SYS 49500 :REM ACTIVATE WEDGE (C64)
10 !I2CAW,112    Start I2C communication with slave 112 (=0x70) for writing
20 !I2CPU,10     Send the value 10 to the I2C device
21 !I2CPU,8
22 !I2CPU,9
23 !I2CPU,1
24 !I2CPU,5
25 !I2CPU,4
26 !I2CPU,6
27 !I2CPU,2
60 !I2CST        Send I2C stop condition, this ends the communication with the I2C slave
70 goto 10
```

Because the PCF8574 is bidirectional, we can also read from it. We can do this as shown below:

```
5 SYS 49500 :REM ACTIVATE WEDGE (C64)
10 !I2CAR,112    Start I2C communication with slave 112 (=0x70) for reading
20 PRINT !I2CGL  Read byte from device and indicate that this is the last byte
30 !I2CST        Send I2C stop condition, ends the communication with the I2C slave
```

The example above shows reading of a single byte, but we may read as many times as we want from the PCF8574. How do we do that, simple, just keep on reading but don't tell the device that it is the last byte until we are really done:

```
5 SYS 49500 :REM ACTIVATE WEDGE (C64)
10 !I2CAR,112    Start I2C communication with slave 112 (=0x70) for reading
20 PRINT !I2CGE  Read byte from device
30 PRINT !I2CGE  Read byte from device
40 PRINT !I2CGE  Read byte from device
50 PRINT !I2CGL  Read byte from device and indicate that this is the last byte
60 !I2CST        Send I2C stop condition, ends the communication with the I2C slave
```


Note:

The PCF8574 is a very simple IO expander, it will switch it's pins to input once there is a READ from the device and it will switch the pins to output when there is a WRITE to the device. So if you intend to input and output multiple signals using the same IO expander then you might get into trouble. In that case use 2 different IO expanders, one for input and one for output. Or choose a better IO expander, there are many to choose from.

3.4.16 Command: ServoController commands

These 2 command are not really integrated in the Cassiopei, they are only a shortcut to control an external device, this is done to keep the overhead as low as possible for anyone who wishes to use the PCA9685 PWM controller for controlling servo's also see the chapter about the expansion connector for more information.

Init servo controller CPIO_SERVOINIT										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	1	0	0	1	0	0	0	1
2	Master write	data	I2C address of servo controller							

Servo positioning CPIO_SERVOPOS										
Byte	Data direction	Byte type	Information							
1	Master write	mode byte	1	0	0	1	0	0	1	0
2	Master write	data	I2C address of servo controller							
3	Master write	data	Servo channel 0-15							
4	Master write	data	Position offset (it is best to set this value to 128)							
5	Master write	data	Position 0-255, if servo perfect and offset=128, then 128=center position							

How to use the servo controller command using the Cassiopei BASIC wedge (C64):

Connect the PCA9685 servocontroller board to the Cassiopei's expansion connector.
Load the cassiopei BASIC wedge into you CBM computer and start it.

Example:

```
5 SYS 49500 :REM ACTIVATE WEDGE (C64)
10 AD=128 :REM SERVO CONTROLLER ON I2C-ADDRESS 128
20 !SERIN,AD:REM INITIALIZE THE SERVO CONTROLLER
30 OF=128:REM OFFSET TO IDEAL VALUE
40 PO=128:REM POSITION IS AT CENTER POISTION
50 !SERVO,AD,OF,PO:REM SEND POSITION
```

Note:

When the servo position and offset are chosen to a value that the servo cannot reach, the servo will draw it's maximum current. This is normal behavior but must be avoided.

3.5 Cassiopei filesystem

In order for the Cassiopei to be really functional there has to be some kind of file storage possible. Below is described how this is achieved.

3.5.1 The flash memory's file system

8MByte of flash memory. Solid state memory the size of multiple disks, unimaginable in the 80's! Well it is available today and we're about to use it. Although not difficult, it is not as easy as it seems. The flash chip can erase blocks of 4Kbyte (not smaller). This means that we cannot erase single bytes, we need to erase 4Kbyte at once. These 4Kbyte blocks are on fixed memory address of the flash chip. Also flash chips cannot be erased endlessly there is always the problem of wear and tear so be careful not to write to the same location too much. This seems very annoying, but practically it does not need to be a problem, as long as you design a proper file system around it. I've come up with the following principle.

Writing to the same flash location over and over again is bad, very bad. So a using a file system for a flash chip that is based on a FAT (File Allocation Table) or BAM (Block Allocation Map) is an evil design. Although there are many flash devices in the world that use FAT file systems, for instance USB sticks (a.k.a. pen drives) or SD-cards. And practically, it is no a problem, considering normal everyday use, it most likely will be out of fashion before it starts breaking down. But there is nothing wrong with doing it better and easier (for this application that is).

But how do you manage files on a chip when using no map of the location of the files. Well, you don't really need a map. This flash chip is fast enough (and our PIC micro-controller is fast enough as well) to scan all possible locations. For a 4MByte chip, that means that there are 1000 blocks of 4Kbyte. So if we want an inventory of the contents of our flash chip. Then we “simply” scan the beginning of every 4K block. Considering that scanning of 1000 blocks takes less then 100mSec (a bit slower with debug printing on). Who cares that we do use a conventional file system. Even better our approach does (in fixed file size cases) does not even require to close the file after writing.

When a file is created, the micro-controller (or computer) scans for a block that holds 0xFF in the first 4 bytes. Why 0xFF, well that's because flash needs to be erased before you can use it. By erasing it all bits are set to 1, making an erased byte look like 0xFF. You can clear every bit (make it 0) on a flash chip independently, but setting it (making a bit 1) can only be done in large sets of bits at the same time, hence the 4Kbyte block erase. So we scan for a block that starts with 0xFF at the first 4 bytes. Once we have found such a block we want to claim it for our use. So we must make sure that it is marked as not-empty, we do that by writing a value (between 0x01 and 0xFE) to the 4th byte of the 4KByte block. (Then a scan for an empty block will no longer accept this block as empty). Then we start writing our data to the bytes 5,6,7,etc. then when we want to write the 4097th byte we will pass the 4Kbyte boundary, we need to put the data in another block, so we search for the next free block, we write the 3 byte address to the first 3 bytes of our current block and then we claim the next block by writing 0x00 at byte location 4 of the 4Kbyte block followed by the data. When there is not enough data to fill this block we will never pass the boundary and we will never alter the first 3 bytes of the current block. Meaning that the last block will have the first 3 bytes at 0xFF. Which is the indication that the file stops at this block.

This method of working creates the following situations for the first 4 bytes of each 4KByte block

<div>block definitions</div> <div>how to determine the status of a block of flash memory</div>		
Byte	Function	Remarks
0	Address byte 2	(MSB) of 3 byte address of next block
1	Address byte 1	(...SB) of 3 byte address of next block
2	Address byte 0	(LSB) of 3 byte address of next block (is always 0x00)
3	<u>High-nibble:</u> Computermodel flag <u>Low-nibble:</u> Indicator (a.k.a. File type)	0x00 = data block 0xX1 = first block of system file, 0xX2 = first block of program file, 0xX3 = first block of vocabulary file, etc
4	data	
//	data	
4095	data	

Note: Address byte 0 is always 0x00 so theoretically it doesn't need to be stored.
 Considering that the 4 four bytes of each block hold the information about the file chain and is therefore crucial for the file system in order to operate, it is marked with the color red. This is also the case for the file type examples in this document.

Because the Cassiopei is designed to be used on different computer models, it is not uncommon to have different kind of files on the flash memory of the Cassiopei. Files intended for different computer models. For instance, a game for the C64 will not work on a PET computer. Therefore a flag to indicate for which computertype the file in intended is very usefull. This flag is located at memory location 0x2E in the relevant filetypes (i.e. .PRG and .TAP). The video type PAL/50Hz or NTSC/60Hz is not indicated.

<div>Computermodel flag</div> <div>a method to indicate the computer model for which the file is intended</div>		
Value	Computermodel	Remarks
0	C64	
1	VIC20	
2	Plus 4	
3	C16	
4	C128	
5	PET/CBM 2000 series	
6	PET/CBM 3000 series	
7	PET/CBM 4000 series	
8	PET/CBM 8000 series	

possibilities of the first 3 bytes of each block

Byte 0	Byte 1	Byte 2	Byte 3	Block type	Remarks
XX	XX	XX	00	Data block	This block is followed by another block starting at memory address indicated by byte 0 (MSB) and byte 1
XX	XX	XX	01-FE	File info and data block	This block is followed by another block starting at memory address indicated by byte 0 (MSB) and byte 1
FF	FF	FF	00	Data block	Last block in the chain (the file ends somewhere in this block)
FF	FF	FF	01-FE	File info and data block	First and Last block in the chain (the file ends somewhere in this block)
0	0	0	1	System file	System file

For the ones who wonder about fragmenting. Yes, this file system like most other file systems can be fragmented as well. However because this is solid state storage, fragmenting does nothing with performance. It does not matter if the blocks are placed next to each other or a scattered around the flash chip. Although it doesn't look nice to humans, fragmentation is nothing to worry about.

3.5.2 System files and user files

The flash memory can hold many file types which can be divided in two kinds of file:

System file:

A system file is expected to be stored at the beginning of the flash memory, meaning 0x000000.

Attention:

There can only be one system file stored on the flash memory!

The size of this file is undetermined and can be up to megabytes in size. Although it is recommended to keep this file as small as possible as it uses up precious user file space.

This file is mainly large tables that cannot be fitted in the micro controllers program memory. Because system files need to be processed as fast as possible they are written different to the flash memory than non system files. To keep programming simple, system files are not block based files but files that need to be written to a continuous memory area. This has the main advantage the exact address location of each byte in the file is known precisely. This in contrast to the block based files which can be scattered all over the flash memory. But since systems files are written to a continuous range of memory address it is very easy to calculate the byte offset in advance.

This advantage becomes clear in the SpeechSample file used for Phonemes (and diphones as well).

User files:

These files are block based and can be added and removed without erasing the entire flash memory. Block based files can be scattered all over the flash memory. They do not need to be written to a continuous block of flash memory. An example of a user file is the .PRG or .TAP file (a game or program to be executed by the CBM computer). There may be many user files stored on the flash memory.

3.5.3 Speech sample file

Speech Sample file (.SYS = filetype 0x1) definition of system file holding speech sample information		
Byte	Function	Remarks
0x00	0x00	The SpeechSample file starts with 3 bytes for compatibility with the filesystem routines. 3 zeroes means that the next block is this block (because a system file always starts at : 0)
0x01	0x00	
0x02	0x00	
0x03	Indicator byte (a.k.a. File type byte)	0x01 = first block of system file Actually the system file is not blocked. But by defining this byte we make it possible for the file system to recognize the file and show it to the user using the standard file system routines.
0x04	File name (ASCII) byte 1	The name which is shown when the user loads the directory of files stored on the flash memory. Empty places are to be filled with 0x20=space
//		
0x23	File name (ASCII) byte 32	
0x24	Filesize MSB	rounded to a 4K boundary value. This is very useful as it indicates the address of the first block after the SpeechSample file.
0x25	Filesize ...SB	
0x26	Filesize LSB	
0x27	Timestamp YY..	example: 20 for the year 2012
0x28	Timestamp ..YY	example: 12 for the year 2012
0x29	Timestamp Month	
0x2A	Timestamp Day	
0x2B	Timestamp Hour	
0x2C	Timestamp Minutes	
0x2D	<undefined>	<<for future use>>
0x2E	<undefined>	<<for future use>>
0x2F	<undefined>	<<for future use>>
0x30	Address Most significant Byte of Sample#1	The SpeechSample file consists of a table that holds the absolute addresses of each sample in the file, that's why this file needs to be stored inside a continuous address range.
0x31	Address Medium significant Byte of Sample#1	
0x32	Address Least significant Byte of Sample#1	
0x33	Address Most significant Byte of Sample#2	This is the absolute address of the first byte of sample#2. It also indicates the end of sample#1. But for ease of programming all samples are also ended with 0x00
0x34	Address Medium significant Byte of Sample#2	
0x35	Address Least significant Byte of Sample#2	
..+0	First byte of sample#1	Data 0x01 - 0xFF
..+1	Second byte of sample#1	Data 0x01 - 0xFF
..+2	Third byte of sample#1	Data 0x01 - 0xFF
..	Last byte of sample#1	The last byte of a sample is always 0x00 to indicate: end-of-sample
..	First byte of sample#2	Data 0x01 - 0xFF
//		

The last byte of of each sample in the SpeechSample file is 0x00. This value is useful as an end of sample indicator, simply because it is easier to check on byte then a 3 byte address. This generates the requirement for the program that creates the SpeechSample file that it filters out the value 0 and replaces it by 1 (which is nearly 0). An easy task, but it must be kept in mind!

Because system files require to be stored in a continuous memory area, it is required to clear the entire system files area before writing a new file. Where as block based files can be deleted and saved independently.

3.5.4 EEPROM file

This file is not a system file but it is not really a user file, it is managed by the cassiopei without the users knowledge.

The Cassiopei needs to store settings data. These settings are possibly changed very often. This means wear and tear on the Flash memory registers. So we need to do some form of wear leveling. But also we must keep in mind that Flash operates in blocks. We can erase a block to FF and write 0's. But if we want to change a single byte in a flash chip we must erase the entire block (which consists of 4096 bytes). Because it is impossible for the PIC to buffer the entire block and because we need to do some form of wear leveling to protect the system from damage due to overactive register writing of some kind.

So the following setup was chosen. Each settings requires a full block, this looks like a lot of overkill and frankly it is. But it keeps programming easy and straight forward. Because of this overhead it is advised to keep the usage of this function low. Simply because it eats up too much memory which could be used for storing games and programs. The Cassiopei only needs a few settings to be saved, MODE (fast, slow, USB, .TAP), INDEX (the index of the program/TAP to be loaded), computer model and the audio feedback volume. These values are stored in the following locations:

Settings locations		
Register	Function	Information
0	Current file mode	Indicates the mode of operation: SLOW, FAST, FAST USB, TAP
1	Current file index	Indicates the index of the file to be loaded
2	Computer model	Indicates computer model (important for correct signal timing)
3	Feedback volume	Indicates volume level of audio feedback of tape signals
4	C128 mode	Indicates if the C128 should work in C128 or C64 mode

The structure is setup to be like a file, this way it can be recognized and used for calculation of the remaining space.

file structure (.EEP = filetype 0xF)
the way a .EEP (EEPROM simulation file) is defined

Byte	Information	Remarks
0x00	Address byte 2	These 3 bytes are all 0xFF This is because there is no next block, the .EEP files are always one block large
0x01	Address byte 1	
0x02	Address byte 0	
0x03	0x20	0x20 indicates a .EEP file
0x04	File name (ASCII) byte 1	The name which is shown when the user loads the directory of files stored on the flash memory. Empty places are to be filled with 0x20=space
//		
0x23	File name (ASCII) byte 32	
0x24	Filesize MSB	0x00 This value is always the same, as this file is only one block large
0x25	Filesize ...SB	0x10 This value is always the same, as this file is only one block large
0x26	Filesize LSB	0x00 This value is always the same, as this file is only one block large
0x27	Timestamp YY..	Not applicable
0x28	Timestamp ..YY	Not applicable
0x29	Timestamp Month	Not applicable
0x2A	Timestamp Day	Not applicable
0x2B	Timestamp Hour	Not applicable
0x2C	Timestamp Minutes	Not applicable
0x2D	<undefined>	<<for future use>>
0x2E	<undefined>	<<for future use>>
0x2F	<undefined>	<<for future use>>
0x30	Data state indicator	Indicates data state: 0x00 – 0xFE=value may be read, 0xFF=do not read
0x31	Data byte 0	
0x32	Data byte 1	
0x33	Data byte 2	

0x40	Data state indicator	Indicates data state: 0x00 = data may be read, 0xFF = do not use for reading
0x41	Data byte 0	

The data is stored in chunks of 16 bytes.

The first byte is: 0x00 to indicate that the next 15 bytes are readable
 0xFF to indicate that the next 15 bytes are not yet written

The last line before a 0xFF line is the most actual data. And that is the data that should be used. The Cassiopei operates in the following manor. When a byte is to be written to a specified EEPROM register all the other values are copied from the previous values and are stored to a new line. When a block is full, it will be erased in order to re-use it again and again. This method lengthens the endurance of the flash memory with 253 times compared to writing the same flash memory cell over and over again.

0x30	00	43	67	76	78	43	dc	3d	5b	a2	22	11	fe	ee	ea	1a	old data
0x40	00	43	67	76	78	43	dc	3d	5b	00	22	11	fe	ee	ea	1a	old data
0x50	00	43	67	11	22	43	dc	3d	5b	00	22	11	fe	ee	ea	1a	data
0x60	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	unused
0x70	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	unused

3.5.5 Vocabulary file

The vocabulary file is a user file, it is defined as:

<word to speak><space><... bytes to describe all phonemes> <0 indicate end of entry in this file>

Vocabulary file (.VOC = filetype 0x3) a collection of pronouncements of words		
Byte	Function	Remarks
0x00	Address byte 2	(MSB) of 3 byte address of next block
0x01	Address byte 1	(...SB) of 3 byte address of next block
0x02	Address byte 0	(LSB) of 3 byte address of next block (is always 0x00)
0x03	0x03	0x03 for the first block, 0x00 for every other block of file
0x04	File name (ASCII) byte 1	The name which is shown when the user loads the directory of files stored on the flash memory. Empty places are to be filled with 0x20=space
//		
0x23	File name (ASCII) byte 32	
0x24	Filesize MSB	rounded to a 4K boundary value
0x25	Filesize ...SB	
0x26	Filesize LSB	
0x27	Timestamp YY..	example: 20 for the year 2012
0x28	Timestamp ..YY	example: 12 for the year 2012
0x29	Timestamp Month	
0x2A	Timestamp Day	
0x2B	Timestamp Hour	
0x2C	Timestamp Minutes	
0x2D	<undefined>	<<for future use>>
0x2E	<undefined>	<<for future use>>
0x2F	<undefined>	<<for future use>>
0x30	<dummy>	Required for alignment, the value is ignored
0x31	<dummy>	Required for alignment, the value is ignored
0x32	T	The word to be spoken
0x33	E	
0x34	S	
0x35	T	
0x36	<space>	Space (0x20) is the separator between word and list of phonemes
0x37	Phoneme ref.	Value between 0x01 – 0xFF that indicates the phoneme to be used
0x38	Phoneme ref.	
0x39	Phoneme ref.	
0x3A	Phoneme ref.	
0x3B	<CR>	Carriage return indicates end of line
...	End of file indicator	Always 0xFF

3.5.6 Program file

First we must keep in mind that the standard CPIO loader program works as follows:

The start address or LOAD address is easy to understand, simply because this determines where to store the program in the C64's RAM. During loading of the program the RAM address is incremented with each byte loaded, once the last address has been reached the loader stops loading and performs a RUN instruction. Which causes the loaded program to execute.

So now it is clear to see that we at least need: last address, start address and file data. A file name to identify the files that are stored and additional information. By storing the information in this order in flash memory we keep the loader program simpler to read. Resulting in a file structure that looks like:

file structure (.PRG = filetype 0x2) the way a .PRG file is defined		
Byte	Information	Remarks
0x00	Address byte 2	(MSB) of 3 byte address of next block
0x01	Address byte 1	(...SB) of 3 byte address of next block
0x02	Address byte 0	(LSB) of 3 byte address of next block (is always 0x00)
0x03	0x02	0x02 for the first block, 0x00 for every other block of file
0x04	File name (ASCII) byte 1	The name which is shown when the user loads the directory of files stored on the flash memory. Empty places are to be filled with 0x20=space
//		
0x23	File name (ASCII) byte 32	
0x24	Filesize MSB	rounded to a 4K boundary value
0x25	Filesize ...SB	
0x26	Filesize LSB	
0x27	Timestamp YY..	example: 20 for the year 2012
0x28	Timestamp ..YY	example: 12 for the year 2012
0x29	Timestamp Month	
0x2A	Timestamp Day	
0x2B	Timestamp Hour	
0x2C	Timestamp Minutes	
0x2D	<undefined>	<<for future use>>
0x2E	<undefined>	<<for future use>>
0x2F	<undefined>	<<for future use>>
0x30	Low byte of last addr. of program in C64 RAM	
0x31	High byte of last addr. of program in C64 RAM	
0x32	Low byte LOAD addr. of program in C64 RAM	
0x33	High byte LOAD addr. of program in C64 RAM	
0x34	Remaining file data	

Items written in **red** are added to the start of each block (where a block consists of 4096 bytes).

Items written in **blue** are added to the .PRG file as found on the PC when programmed to flash.

Items written in black are copied directly from the .PRG file as found on the PC.

3.5.7 TAP file

A .TAP file is nothing more than a stream of data. That stream must be sent to the C64 as if it was an audio signal. Which in practice is nothing more than setting a pin high or low with an accurate timing.

file structure (.TAP = filetype 0x4) the way a .TAP file is defined		
Byte	Information	Remarks
0x00	Address byte 2	(MSB) of 3 byte address of next block
0x01	Address byte 1	(...SB) of 3 byte address of next block
0x02	Address byte 0	(LSB) of 3 byte address of next block (is always 0x00)
0x03	0x04	0x04 for the first block, 0x00 for every other block of file
0x04	File name (ASCII) byte 1	The name which is shown when the user loads the directory of files stored on the flash memory. Empty places are to be filled with 0x20=space
//		
0x23	File name (ASCII) byte 32	
0x24	Filesize MSB	rounded to a 4K boundary value
0x25	Filesize ...SB	
0x26	Filesize LSB	
0x27	Timestamp YY..	example: 20 for the year 2012
0x28	Timestamp ..YY	example: 12 for the year 2012
0x29	Timestamp Month	
0x2A	Timestamp Day	
0x2B	Timestamp Hour	
0x2C	Timestamp Minutes	
0x2D	TAP file version indicator	0=TAP file V0, 1=TAP file V1, 2=TAP file V2
0x2E	Compression type	1=no compression, C0=compression V0
0x2F	Compression escape code	The value of the escape code used
0x30	Uncompressed filesize MSB	The size of the data of the TAP file, this is required because otherwise it is impossible to determine the size of the file without completely reading it. Knowing the size of the file is required (for example) for the tape positioning slider.
0x31	Uncompressed filesize ...SB	
0x32	Uncompressed filesize LSB	
0x33	?	
0x34	Remaining file data	

Items written in *red* are added to the start of each block (where a block consists of 4096 bytes).

Items written in *blue* are added to the .TAP file as found on the PC when programmed to flash.

Items written in *black* are copied directly from the .TAP file as found on the PC.

TAP file compression

TAP files are known to be large, very large compared to the information they hold. A .TAP file size is generally between 200kByte – 2MByte. The reason why these files are so big is because the method the data is stored on the audio tape. A .TAP file is an image of the contents of an Commodore computer data tape. The bits are described as a sequence of sinus-shape waveforms (actually the computer writes them as blocks but due to the limitations of the tape and electronics this is reduced to something that looks more like a sinus than a block. The data is stored in the frequency of the signal and not the volume. So digitizing the data can be done by measuring the period of the signal. When these are known then the original block signal can be recreated.

A single byte is described as a series of bits, and with each bit being a sinus and with some overhead in indicating the start of a byte, this will result in a lot of sinus shapes to describe a byte. And each sinus is described as a single byte value, so in practice this would mean approx.10 bytes of data to describe a single byte. But that is not all, some .TAP files are longer than they should be, because the silence is digitized as well, but silence is never completely still, so the cracks and ticks of the tape result in fake data, fortunately this is not a big issue because most .TAP files are cleaned up before they are put online.

It is not possible to interpret/read the data from a TAP file, because this would require great knowledge of the loader used. Because in the good old days the Commodore kernel loader was abandoned for a better (custom made) loader for game, it could load faster and show an image of the game and play music while loading, wow, that was great. But many kinds of these fastloaders existed so there are simply too many variations that must be implemented in order to fully compress the data. But there is another solution. It is called compression.

Before you think of .ZIP, .RAR, etc. you must realize that the compression algorithm cannot be too complicated, because we must run it in “realtime” on a microcontroller with very little RAM. And because most well known compression algorithms use huge dictionaries, we have to find a compromise.

Fortunately the data in .Tap files is highly repetitive. So the trick is to find the parts of the data that are repeated over and over again. By describing these sequences of repeated data the file can be made smaller. For example, if we have the following sequence of bytes:

```
22 00 AA A3 AA FE F0 14 14 14 14 14 11 64 64 64 64 64 00 AA A3 AA FE F0 12
```

Here we can see that the sequence 00 AA A3 AA FE F0 is repeated, so we can write in a compressed version by pointing to the location where we've used the same sequence before:

```
22 00 AA A3 AA FE F0 14 14 14 14 14 11 64 64 64 64 64 <esc><17><6>
```

meaning “use here a sequence of 6 bytes that you can find 17 bytes back”

This sequence of 6 bytes now only takes 3 bytes to describe. So we've made the file smaller. This of course is only useful for sequences that are 4 bytes or longer. But due to the repetitive nature of .TAP files this is very usable.

The <esc> is a code used to indicate the pointer to the data. Or in other words, when you find this code during decoding you must use the next 2 bytes as a pointer to data and not as data. But the data in a .TAP file can vary between 00 and FF so what code should we use for an escape code. There are techniques like byte stuffing that make it possible to create unique codes especially for this purpose but using this technique would mean extra overhead and therefore a lower compression ratio. Fortunately when I analyzed many .TAP files I discovered that there are values between 00 and FF that are not

found in the .TAP file. This differs for all .TAP files, but potentially there are enough values suitable for an escape code without byte stuffing. And we need only 1 value to define the escape code.

At the beginning of the compression of the .TAP file the file is scanned for a suitable escape code. Meaning that the entire file is scanned for unused codes and the highest value will be used as the escape code. This value depends on the .TAP file and therefore may vary. But by storing the value of the escape code in a special field in the header of the file as stored onto the Cassiopei, the Cassiopei will know the value of the escape code.

As you can see the pointer used is only 2 bytes, 1 for the location in the data array and 1 for the length of the sequence. 1 byte would mean a max. range of 256 bytes. So what the decoder needs to do is to remember the last 256 bytes of the decoded data. This way if a sequence of bytes is repeated within this range, then it is compressible. And 256 bytes is the perfect size for the microcontroller inside the Cassiopei. Now you might think that you can play with the pointer sizes. Because these are 16 bits (8 for pointer and 8 for length), what if we made the pointer 9 bytes and the length 7, then we have a larger area to search in but we cannot handle sequences longer than 128 bytes. Technically this is all possible, but changing the pointer from 8 to 10 bits (quadrupling the microcontrollers RAM memory requirements) resulted only in a (max.) 5% gain of the compression factor. Considering the ease of programming on both sides (microcontroller and PC) the 8 bits pointer method was chosen. Which effectively compresses the average .TAP file by a factor of more than 70% (a file of 1Mbyte would then only be 307KByte). This is still a large file but much more acceptable. Considering that the Cassiopei has only 8Mbytes of flash memory available. So on average, this simple method of compression allows you to store more than 3 times as much .TAP files onto your Cassiopei.

Now this is not all there is to compression, because the above situation isn't all that practical when it comes to random file access. Imagine that you want to start reading from a compressed file from the uncompressed location 234320 and further. Then this would be a problem, because the data is compressed, the codebook/dictionary/history of the last ... bytes used is dynamic. The only way to get to the desired point is by also reading all the previous bytes in order to get the correct codebook status. Although this is not a problem for small files it is for a filesize of megabytes, which would take up many seconds to reach the desired location. This is unacceptable for a modern device. Therefore we need to build in sync. points. A point at a fixed file location that resets the status of the codebook so that we can read from that point without having to read all the previous bytes first. To let the reader of this compressed file with sync. points know where he/she is in the uncompressed file the sync. point holds a 3 byte value that indicates the position of the uncompressed file. Now it is possible to wind and rewind much faster than reading always from the beginning of the file. Go to the first sync. point, read the fileposition value, if this is smaller than the desired position, then go to the next sync. point and repeat this until the sync. point is found that is larger than the desired position, go back to the previous sync. point and read from there byte for byte until the desired sync. point has been reached. With sync. points located exactly at a distance of 16Kbytes (16384Bytes) apart, it will not affect the compression ratio but allows (from a user perspective) instantaneous winding and rewinding.

The interpreter/reader of this compressed file must be able to ignore the sync. points or in other words prevent the sync. points from being seen as file data. But also the resetting of the codebook must be respected when the sync. point is reached.

The location of the sync. points is counted from the start of the payload (i.o.w. the actual tape data). A visual representation of the location of the sync. points is shown in the image on the nextpage.

Sync. points in a compressed TAP-file

Absolute loc. in file on flash	Function	
0x000000	Start of file	Header information
0x000034	Sync. point	MSB of fileposition in uncompressed file
		MSB of fileposition in uncompressed file
		LSB of fileposition in uncompressed file
		Compressed data of TAP file
		Compressed data of TAP file
		...
		Compressed data of TAP file
0x004034	Sync. point:	MSB of fileposition in uncompressed file
		MSB of fileposition in uncompressed file
		LSB of fileposition in uncompressed file
		Compressed data of TAP file
		Compressed data of TAP file
		...
		Compressed data of TAP file
		Compressed data of TAP file
0x008034	Sync. point:	MSB of fileposition in uncompressed file
		MSB of fileposition in uncompressed file
		LSB of fileposition in uncompressed file
		Compressed data of TAP file
		Compressed data of TAP file
		...
...
0x05C034	Sync. point:	0xFF the last sync. point in a file is indicated by the max value
		0xFF this is to prevent reading past end-of-file
		0xFF
		Compressed data of TAP file
		Compressed data of TAP file
		...
		Compressed data of TAP file
		End of file

3.5.8 WAV file

A .WAV file is nothing more than a stream of data of an 8Khz, 8-bit, mono, audio sample.

file structure (.WAV = filetype 0x5) the way a .WAV file is defined		
Byte	Information	Remarks
0x00	Address byte 2	(MSB) of 3 byte address of next block
0x01	Address byte 1	(...SB) of 3 byte address of next block
0x02	Address byte 0	(LSB) of 3 byte address of next block (is always 0x00)
0x03	0x05	0x05 for the first block, 0x00 for every other block of file
0x04	File name (ASCII) byte 1	The name which is shown when the user loads the directory of files stored on the flash memory. Empty places are to be filled with 0x20=space
//		
0x23	File name (ASCII) byte 32	
0x24	Filesize MSB	rounded to a 4K boundary value
0x25	Filesize ...SB	
0x26	Filesize LSB	
0x27	Timestamp YY..	example: 20 for the year 2012
0x28	Timestamp ..YY	example: 12 for the year 2012
0x29	Timestamp Month	
0x2A	Timestamp Day	
0x2B	Timestamp Hour	
0x2C	Timestamp Minutes	
0x2D	0x80	<<for future use>>
0x2E	0x80	<<for future use>>
0x2F	0x80	<<for future use>>
0x30	0x80	<<for future use>>
0x31	0x80	<<for future use>>
0x32	0x80	<<for future use>>
0x33	0x80	<<for future use>>
0x34	Remaining file data	
0x- - -	0x00	Last byte always zero (end of sample indicator)

Items written in **red** are added to the start of each block (where a block consists of 4096 bytes).
Items written in **blue** are added to the .WAV file as found on the PC when programmed to flash.
Items written in **black** are copied from the .WAV file as found on the PC.

The end marker 0x00 is a unique value, therefore the WAV-file values are scanned before saving to flash. If the value 0x00 is in the sample data then it is changed to 0x01 to prevent a false detection of the end of the sample.

3.5.9 DAT file

A .DAT file can hold anything that can be interesting for a game or program to use. For instance a very large scroll text or the levels/screens for your game. The programmer must be aware that this file holds no end-of-file marker, therefore the program using this file must prevent reading past the end of the file.

file structure (.DAT = filetype 0x6) the way a .DATA file is defined		
Byte	Information	Remarks
0x00	Address byte 2	(MSB) of 3 byte address of next block
0x01	Address byte 1	(...SB) of 3 byte address of next block
0x02	Address byte 0	(LSB) of 3 byte address of next block (is always 0x00)
0x03	0x06	0x06 for the first block, 0x00 for every other block of file
0x04	File name (ASCII) byte 1	The name which is shown when the user loads the directory of files stored on the flash memory. Empty places are to be filled with 0x20=space
//		
0x23	File name (ASCII) byte 32	
0x24	Filesize MSB	rounded to a 4K boundary value
0x25	Filesize ...SB	
0x26	Filesize LSB	
0x27	Timestamp YY..	example: 20 for the year 2012
0x28	Timestamp ..YY	example: 12 for the year 2012
0x29	Timestamp Month	
0x2A	Timestamp Day	
0x2B	Timestamp Hour	
0x2C	Timestamp Minutes	
0x2D	0x80	<<for future use>>
0x2E	0x80	<<for future use>>
0x2F	0x80	<<for future use>>
0x30	Uncompressed filesize MSB	The size of the data field. Because this filetype is general purpose, it can be used for anything. The filesize is useful in situations where a file can't or doesn't have an "end of filemarker".
0x31	Uncompressed filesize ...SB	
0x32	Uncompressed filesize LSB	
0x33	0x80	<<for future use>>
0x34	Remaining file data	
0x- - -	??	Last byte of the file is not indicated by any kind of marker

Items written in *red* are added to the start of each block (where a block consists of 4096 bytes).

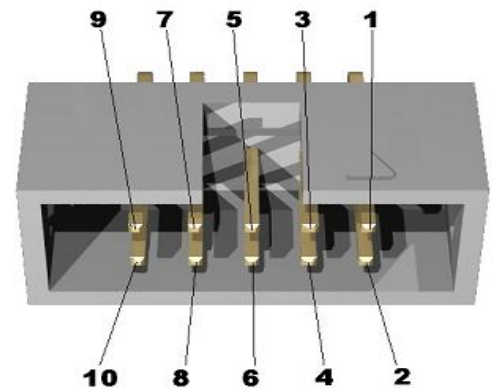
Items written in *blue* are added to the .DAT file as found on the PC when programmed to flash.

Items written in black are copied directly from the .DATA file as found on the PC.

3.6 Expansion connector

This section explains how to connect your own hardware to the expansion port of the Cassiopei. Using the expansion port of the cassiopei for your own designs requires the basic knowledge of electronics. Always work careful and secure but most importantly use your common sense.

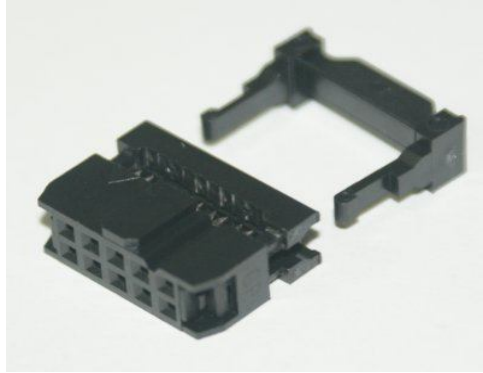
The connector of the expansion port has the pinout as shown in the table below. The IO-pins on this connector can have more then one function. This is possible because of the sophisticated software inside the Cassiopei's microcontroller. Depending on the command you send the required IO-pin will be set into the correct mode of operation. IO-pins that are not required are to be left unconnected.



Expansion connector pinout			
Pin	Data direction	Function	Information
1	Input	Reset	Reset signal of the Cassiopei, directly connected to the reset button. Pressing the button makes this signal low, resetting the cassiopei.
2	Output	PWM	PWM audio output. Requires filtering to be used see paragraph Audio output (PWM)
3	Analog input	Chan. 0	Analog input channel for ADC.
	Output	Klikaan KlikUit	Signal for both 13 bits and 32 bits KlikAanKlikUit signals.
4	I ² C	SDA	Data line of I ² C bus.
5	Analog input	Chan. 1	Analog input channel for ADC.
	Output	reserved	Reserved for future purposes
6	I ² C	SCL	Clock line of I ² C bus.
7	n.a.	+3.3V	Do not use unless your circuit is very low power! Power supply of the Cassiopei. Max power (50mA). This voltage has a tolerance of max. 5%. Verify this voltage using a volt-meter if you intend to do accurate AD conversions.
8	n.a.	GROUND	Ground of the entire system, connect to the ground of your own circuit. Also connect your circuit to pin 10 for best performance.
9	Output	TXD	Reserved for debugging purposes.
10	n.a.	GROUND	Ground of the entire system, connect to the ground of your own circuit. Also connect your circuit to pin 8 for best performance.

3.6.1 Required Cassiopei expansion connector

Below you'll see an image of the connector that plugs into the Cassiopei's expansion connector. This connector is very common and therefore should be easy to find. It is produced by many manufacturers for example 3M (product code: D89110-0131HK),. Most electronic hobby shops will be capable of supplying these connectors, just show them your Cassiopei and they know what connector you'll need.



This connector is also available at www.Conrad.nl product number: 742106-89

Keep in mind that this is only the connector, you must crimp a ribbon-cable to this connector (using a vise), the ribbon cable must have a pitch of 1,27mm.

The cheapest way of finding this connector including ribbon cable is by salvaging them from an old PC, as this cable is used for connecting the serial port connector to the motherboard of the PC, see the image below. **The 9 pole sub-D connector should be removed !** Although most of these cables do not include pin-10. Which is the Cassiopei's ground, however this is also available at pin-8, so it is not a big problem.



Pin 1

On the ribbon cable connector pin-1 is indicated by a small triangle.

On the cable wire-1 is indicated by the red wire of the ribbon cable. Always double check!

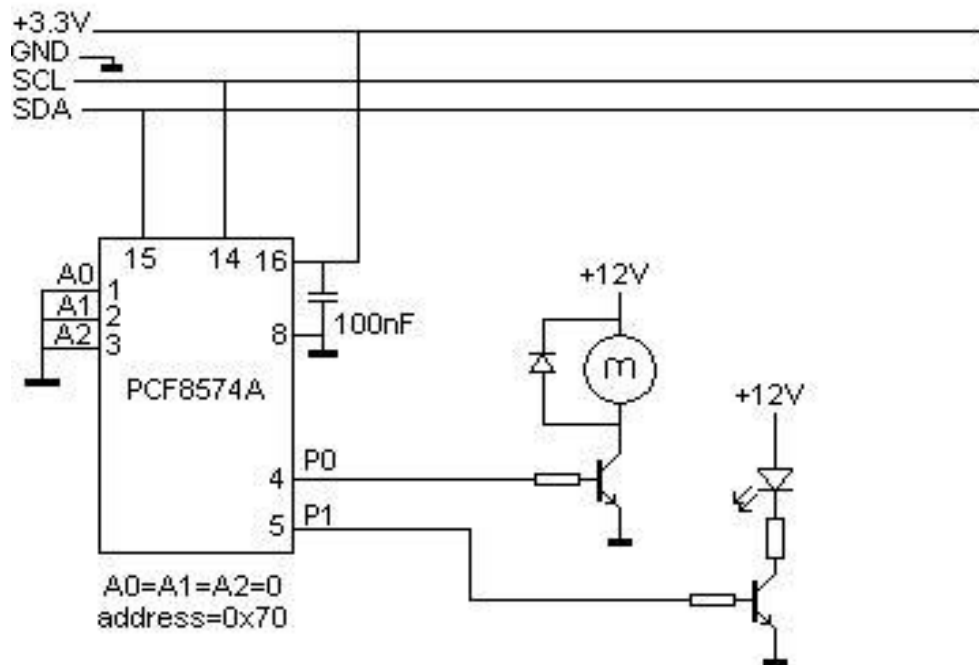
So if the cable is properly connected to the connector, the red wire should meet the triangle marking on the connector.

3.6.2 Voltage levels of the Cassiopei

The Commodore computer operates at 5V internally, the Cassiopei operates at 3.3V internally. The Cassiopei is connected to the Commodore computer using resistors in order to lower the voltage level of the Commodore computer so that it would not damaged the Cassiopei. So you do need not to worry, your Commodore computer is absolutely safe.

However, when you connect your own electronics to the Cassiopei's expansion connector you must be aware that the expansion connector operates at 3.3V. Meaning that everything that you connect must not exceed that value. If you connect a voltage of more then 3.3V to the ADC inputs, you might damage it (for more info see the section about the ADC input).

If you want to use the I2C bus then you must use I2C devices that are fed from a 3.3V power source. If you feed your I2C devices with 5V, then the I2C bus must operate at 5V which is not the case because the Cassiopei's I2C bus is 3.3V. Fortunately most I2C devices can operate at 3.3V. And if you use an IO-expander (for instance the PCF8574A) to drive some motors or lights or relays then you require additional transistors to be able to switch the required currents and by doing so you are able to use higher voltages to switch your own devices. See the example below:



3.6.3 Reset input

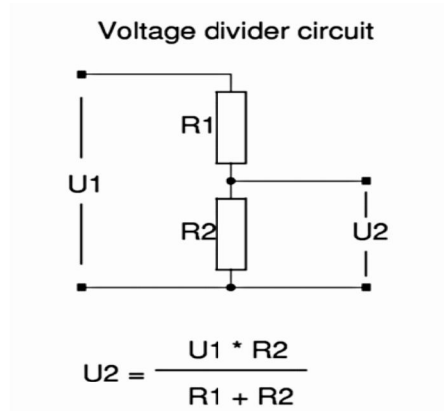
The Cassiopei has a micro-controller which can be forced to reset by making this signal low (connect to ground). This line is pulled up by an on-board 10K resistor, so this line can be left unconnected. This line is also connected to the reset button of the cassiopei. So pressing this button drives this line low. This reset signal can be used for external logic. However caution is required. When using long cables to connect the cassiopei to your own (self designed) circuitry additional filtering or pull-up resistors at the inputs of the reset signal of your own circuit may be required. This because the cassiopei's on-board 10K pull-up resistor is not sufficient to suppress disturbances on the reset line.

Example: you have an MCP23008 IO expander (this one has a rest input) that you connect to the Cassiopei's I2C bus of the cassiopei's expansion port. You connect the rest line to the input of your MCP23008 using a cable with a length of 1 meter. Then you may encounter EMC problems. For instance your circuit (not the cassiopei) could reset “spontaneously” when you switch on the lights or some other noisy electrical device. You can solve this by connecting the reset input of the MCP23008 directly to Vcc (+3.3V) (meaning you no longer use the reset line of the cassiopei to reset your own device). Or you add an additional 1K (not lower) resistor close to the input of your MCP23008. So now you can still use the reset button of the Cassiopei to reset you own electronics.

However, it is preferred not the use the cassiopei's reset line as a signal for your own circuitry. This because this signal is poorly buffered and cannot drive any significant current. Also some electronic device require a more rigid (less bouncy) reset signal. So it is best practice to always design in a reset circuit of your own. Making your own circuit independent of the reset of the cassiopei.

3.6.4 Analog inputs (ADC)

The Cassiopei has on-board Analog to Digital Converters, these are converters that can convert a voltage between 3.3V and 0V (the cassiopei's operating voltage) with a resolution of 10bits. Do not apply higher voltage then 3.3V to the ADC's input pins or damage will occur!!. Higher voltages then 3.3V can be applied by the use of a voltage divider.



U1 is the voltage you want to measure, R1 and R2 determine the scaling, the divide the voltage to a lower level that fits into the range of the cassiopei. U2 is the voltage that is applied to the input of the cassiopei ADC channel. For an accurate measurement R2 may not be higher then 1Kohm. Also make sure that you know exactly what values you put in. Keep in mind that the resistor you put in might be slightly higher or lower due to it's tolerance, so for an accurate measurement use high tolerance resistors (0.1%) or just measure the value of your resistor using a multimeter and adjust the scaling values in the program with the values you measured, which is the cheapest solution. Because high accuracy resistors can be very expensive.

An additional capacitor is always a good idea, add 100nF to the ADC's input pin to suppress noise.

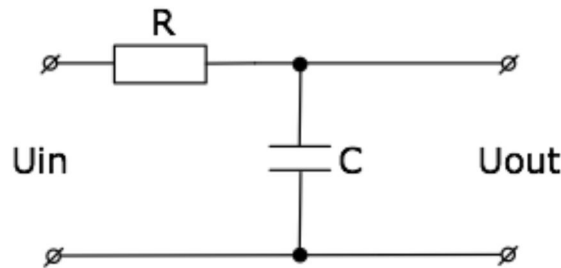
The Cassiopei's power supply may have an error of 5%, this error is constant. In most cases it will be exactly 3.30V but you'll never know unless you check it with a voltmeter. If you know the error then you know the correction you must make. See the BASIC wedge example for the ADC.

Disclaimer:

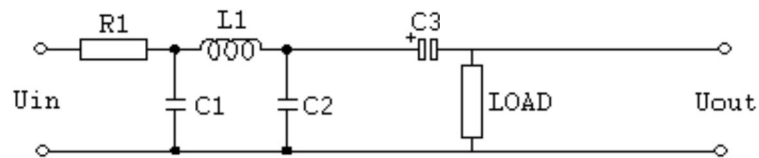
Measuring voltages must always been done carefully and with common sense, do not attempt to measure the mains voltage or other dangerously high voltages using a voltage divider and the cassiopei, this is unsafe and damage will occur !!!

3.6.5 Audio output (PWM)

The audio signal is carried by pin 2 of the expansion connector (pin 8, 10 are GND). This signal is a PWM signal and needs to be filtered using a small filtering circuit as shown below. It cannot be used without filtering. Use $R = 1\text{ K}$ and $C = 470\text{ nF}$.



The filter above is not a real good filter but it smooths out the PWM signal to a usable level. A much better filter would be a Chebyshev filter. Made of R_1 , L_1 , C_1 and C_2 . Where $R_1 = 1\text{ K}$, $C_1 = 68\text{ nF}$, $C_2 = 68\text{ nF}$, $L_1 = 33\text{ mH}$. The load of this filter should be 1 K . The capacitor $C_3 = 100\text{ uF}$ is used for blocking the DC component, although every audio amplifier should have such a capacitor on its input it doesn't harm to put one in the output of our filter. The input of the SID chip is 1 K and there is a DC blocking capacitor on the audio input of the C64. So with this capacitor and the SID matching the load resistor perfectly, this would mean that if you connect the output of the filter to the audio input of the C64 you do not have to solder in a load and you may leave out the capacitor C_3 .



Using one of the above filtering techniques you change the PWM signal from the Cassiopei to a level that matches the input requirements of the SID chip. The input of the SID chip is controlled by the volume of the SID itself. Meaning that after reset the SID will not let you hear the sounds you put onto the input of the SID. Therefore the Cassiopei's BASIC wedge enables the volume to max. level. Otherwise you would not hear anything. Mixing audio signals over the SID chip has the advantage that the audio signal is also outputted over the RF-signal of the RF modulator of your C64. Meaning that when you connect your TV to your C64 over the RF cable, you would hear the sound the Cassiopei produces. But also over the audio out if you connect your TV monitor using the AV cable.

Attention:

The SID chip's input is controlled by the main volume setting of the SID. This means that the SID can modulate the volume of the input. Default this volume is set to 0. In order to make the SID's input audible set the volume to the desired value. To set the SID volume to the maximum level use POKE 54296,15. The setting needs to be taken into account when the PWM signal is fed to the SID but the Cassiopei's BASIC wedge is not used.

3.6.6 I²C bus

The Cassiopei has I²C bus (which may also be referred to as the IIC or I2C bus) that makes it very easy to connect additional circuitry with a minimum of wires. There are many different IC's that feature and I²C interface and as long as they can be used with a working voltage of 3.3V they can be connected to the Cassiopei. The voltage of 3.3V seems low comparing it to the 5V of the CBM computer, but this is a modern device and modern devices run on much lower voltages than the older technology of the 80's. 3.3V is still rather high if you compare it to the circuitry in your PC or mobile phone, but that's a detail. It is no problem at all to find an I²C device that can work on 3.3V. Below is a short list of I²C devices that can be directly connected to the Cassiopei's expansion port. Please download and read their data-sheet on how to use them, this document cannot handle all possible devices. The Cassiopei's I²C bus is designed to send and receive all required I²C commands, by defining the proper sequence of signals for example by using the Cassiopei's BASIC wedge, you can control almost all I²C devices on the market.

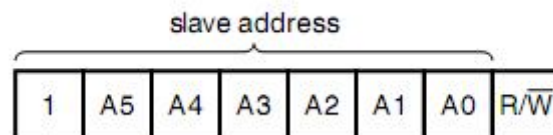
Reserved I²C addresses

Below is a list of I²C addresses that are reserved for Cassiopei functionality. This means that these addresses cannot be used by your own project. Because if you then experience unexpected behavior of your own circuitry (or bus conflicts) will occur.

Reserved I²C addresses (do not use these addresses for your own project)		
Address	Function	Information
0x78	OLED display	An optional display (SSD1306) connected to the expansion port of the Cassiopei can show status information of the Cassiopei. For instance: computer model, operation mode and current file.

Addressing of I²C devices on the I²C bus using the Cassiopei's BASIC wedge:

I²C addressing can be a bit confusing if you see it for the first time. This is because the address has the R/W bit at the LSB location (bit-0). Some devices allow you to configure one or more of the address bits. The PCA9685 even allows you to set 6 bits of the I²C address. Let's assume that you have configured your hardware by connecting the address inputs to Vcc (a logic '1'). So then A5=1, A4=1, A3=1, A2=1, A1=1, A0=0. Then what would the proper definition of the address be?



Because some people ignore the R/W bit when they speak about the device's address so in that case the device's address would be 7 bits and not 8 and it would be 1111111 = 0x7F = 127 which is confusing to say the least! As we send 8 bits over the bus.

So do it right and use all the 8 bits when you speak about the device address. Always assume R/W=0. The I²C routines will fill in the R/W bit depending on the reading or writing action. Then the device must be called using 8-bit address 11111110 = 0xFE = 254.

So please remember to always use the addressing with all the 8 bits including the R/W bit and call the device by its proper address. If you are not sure about the addressing, then check the data-sheet of the device you are using and look for an image as mentioned here.

PC devices you could use for your project

Here are described some of the many available I2C devices that you could use for your own project.

PCF8574 or PCF8574A (the A indicates a different base address of the device).

This is a very well known and widely available IO expander, the PCF8574 has 8 IO pins that are output when you write a value to it and input when you read a value from it. It is as easy as that. Please check the data-sheet of the chip you buy before you connect your circuit as the output pins may require some extra buffering. You can connect up to eight of these devices as they have 3 inputs to change the device address. Using both the PCF8574 and the PCF8574A you can connect up to 16 of these IO expander. This would result in adding $16 \times 8 = 128$ IO pins to your CBM computer. Keep in mind that the PCF8574 has 8 IO pins that are all 8 input OR all 8 output they cannot be mixed. If you require extra input and output in one design, simply use 2 8574's one for the extra inputs and the other for the outputs.

MCP23008 (8bit IO)

This device is similar to the PCF8574, but it has much more functions, you can mix inputs and outputs, even invert the data if required, interrupts etc. This makes the chip a little bit more complicated to operate, but the data-sheet explains it all, so download the data-sheet and start reading...

MCP79410 or MCP79411 or MCP79412 (real time clock)

A real time clock that counts the time independently of your CBM so the time is always correct and you never have to set the date and time again as this device keeps track of it. All you've got to do is write the software for it.

PCA9685 (12 bit PWM controller (use it as a servo controller to drive 16 standard RC servos))

This device is great for controlling RC hobby servo motors. Although it is a chip designed for controlling LED's of televisions (think of the Philips ambilight televisions) it can do much more. It can generate a PWM signal of with an accuracy of 12 bits. And by settings the registers of this device to the proper timing values, you can very easily generate the signal to drive a RC servo motor to the desired position. And because this device has 16 channels, you can control up to 16 servo's with 1 device. You can connect up to 62 of these devices onto the I2C bus ($62 \times 16 = 992$, that are a lot of servo's)!

RC hobby servo controller using the PCA9685

The Cassiopei's I²C bus can connect to the PCA9685 I2C 16 channel PWM controller. And by configuring it correctly it can generate the signals required for driving a RC hobby servo motor.

A servo signal consists of a pulse of minimal 1 mSec and a period of (approximately) 20 mSec. The position of the servo is determined by the size of the pulse. 1 mSec positions the servo to the minimal position (which could be left or right depending on the used servo). And 2 mSec positions the servo to the maximal position (which could be right or left depending on the used servo). But this is not really true. As the max swing of a servo is 180deg, while a pulse of 1-2mSec only moves it 90deg. So there is extra movement of 45deg on both directions. This is done to create some room for adjustment in the RC model, which is required most of the times. It allows you to compensate for tolerances or to “fine-tune”. This tuning possibility is called offset, it is a fixed value added to the position. But there is nothing wrong if you play with this value in combination with the position, this effectively doubles the range of the servo. Please beware of not moving the servo to much to it's limits, otherwise it will draw too much current.

It is very important that the positioning pulse is accurately timed. A jitter on this signal causes a jitter in positioning. Although this is not always noticeable in movement, it is audible (the motor humms). So a stable signal creates stable positioning. A micro-controller generating servo signals using interrupts has the jitter problem and there is not much you can do about it to fully eliminate that. PWM signals are created by dedicated hardware that does not suffer from that problem. But most micro-controllers do not have the PWM generator with the proper resolution to create these servo signals. The PCA9685 does have the resolution. And it has 16 channels, that's more then most people ever need. Regarding timing of the servo pulse there is also the period, this may vary, it's not crucial but it is preferred to have it stable as well (without jitter). Using the PCA9685 this is also no problem. The repetition rate of this signal is best set to 50Hz.

The PCA9685 must be configured properly in order to function, the registers need to be filled with the following values (the registers as shown below must be configured in the specified order):

Configure PCA9685 as a servo pulse generator		
Register	Function	Information
0	MODE-1	16 : Set controller into sleep mode in order to adjust pre-scaler
254	PRE_SCALE	121 : Adjust pre-scaler
0	MODE-1	32 : Enable controller and set it into address incrementing mode
6	Output-0 ON low byte	Set to 0 because we want our servo pulse to start at the beginning of the period of this signal.
7	Output-0 ON high byte	
8	Output-0 OFF low byte	Set to the position 65 + offset + position (105 creates a 0.5mSec pulse) offset = 0-255, position =0-255 (both values determine the position)
9	Output-0 OFF high byte	

How to use the PCA9685 with the cassiopei BASIC wedge:

Load the cassiopei BASIC wedge into you CBM computer and start it.

You can configure and program the PCA9685 yourself but it doesn't make your code any easier as it requires a lot of overhead in reading and writing registers of the IC and calculating counter values. Fortunately the cassiopei BASIC wedge implements the PCA9685 in a very practical manor. With only 2 commands you can drive your servo's to any desired position.

!SERIN, A

A is the I2C address of the PCA9685. This command initializes the device to the proper settings in order to generate the servo signals.

!SERVO,A,C,O,P

A (2-254) is the I2C address of the PCA9685.

C (0-15) is the channel we want to control. The PCA9685 has 16 channels

O (0-255) is the offset (0=max offset, 128=suggested value 255=max offset,)

P (0-255) is the position

Example (C64):

```
5 SYS 49500: REM START THE WEDGE (C64)
10 REM DRIVE SERVO MOTOR BETWEEN 2 POSITIONS
20 A=254 :REM ADDRESS
21 !SERIN, A:REM INIT SERVO CONTROLLER
22 C=0 :REM SERVO IS CONNECTED TO CHANNEL-0
23 O=128 :REM OFFSET IS NONE
30 P=0 : !SERVO,A,C,O,P
40 FOR L=0 TO 1000:NEXT L
50 P=250 : !SERVO,A,C,O,P
60 FOR L=0 TO 1000:NEXT L
70 GOTO 30
```

3.6.7 KlikAanKlikUit signals

“KlikAanKlikUit” is a 433.92 MHz remote control system for switching lamps and everything else you can plug into a power outlet. It is sold in the Netherlands (by the GAMMA). This system exist for many years and has a wide range of devices that can be used. Today (2013) there are 2 generation of this system.

The first generation, which is 13 bits and requires the user to use to set on his remote control and receivers slide/rotary switches to define the device ID.

The second generation, is a 32bits system that is capable of “learning” the codes of your remote control, there are no rotary switches on the receiver.

Although it is stated that you could use the first generation of remote controllers with the second generation of receivers, I would not recommend to do so. For more information about compatibility between remote controllers and receivers visit <http://www.klikaanklikuit.nl>

For the Cassiopei, it does not matter. The Cassiopei is capable of generating both first and second generation of codes. All that you need to do it to connect a 433.92 MHz transmitter to your Cassiopei's expansion port. Now where do you get a 433.92 MHz transmitter from you might ask. Well the easiest way would be to modify an existing KlikAanKlikUit remote control. This can be easily done if you have the basic electronic soldering skills. You can modify a remote of the first or the second generation, it does not matter, they are both sending at exactly the same frequency of 433.92MHz and we are only using the RF stage of the remote control. On the next few pages are the descriptions on how to modify the PA3-1000R and the AYCR-102 remote control.

How to connect a KlikAanKlikUit PA3-1000R remote control to your cassiopei.

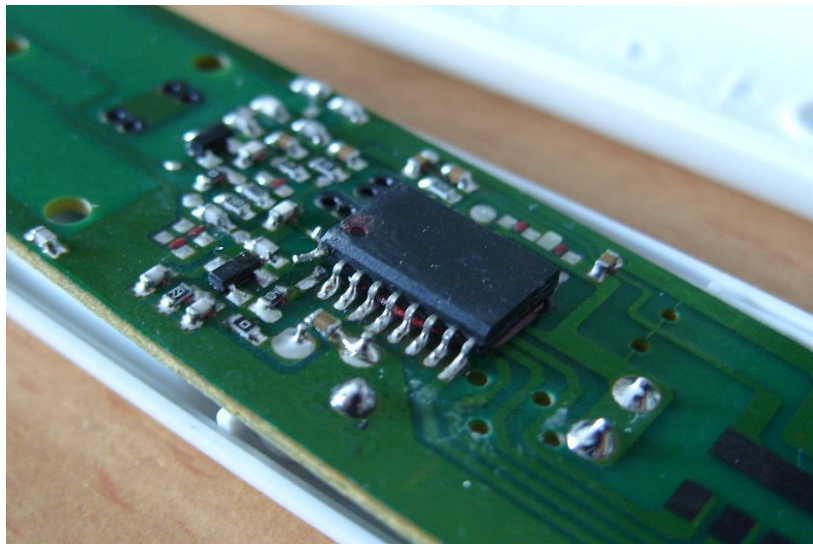
This is how the modified PA3-1000R remote control will look like when the mod is complete. As you can see, it plugs directly into your cassiopei's expansion connector. This modification no longer allows the use of the buttons, the Cassiopei (or to be more precise, the CBM computer will be in full control).



Remove the battery of the PA3-1000R remote control and using a Philips screwdriver open the remote control. Store the battery in a safe place, we won't need this any more. The cassiopei's power supply is sufficient to drive the remote control.



Using a soldering iron and some pointy tweezers, gently lift pin 1 of the IC on the PA3-1000R remote control PCB. The IC is now no longer connected to the RF circuit.



10 pole ribbon cable

Pin 1 = not used (this wire is indicated by the RED stripe on the cable)

Pin 2 = not used

Pin 3 = carries the KlikAanKlikUit signal connect this to the input of the RF circuit of the remote

Pin 4 = not used

Pin 5 = not used

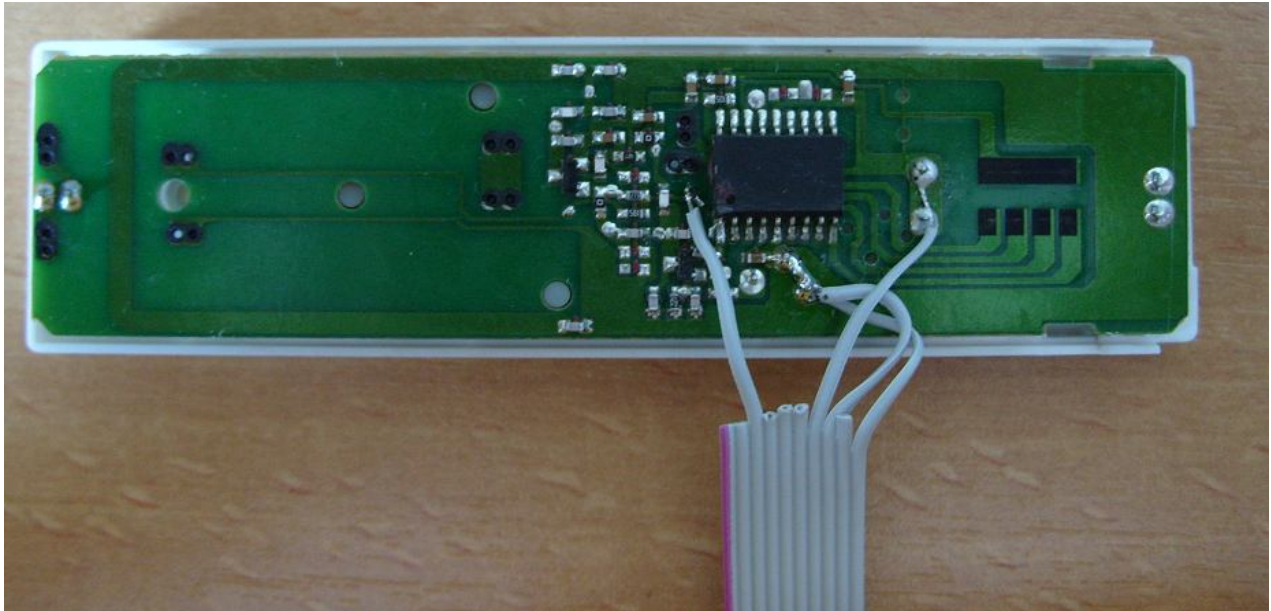
Pin 6 = not used

Pin 7 = +3.3V, connect to the battery + pad

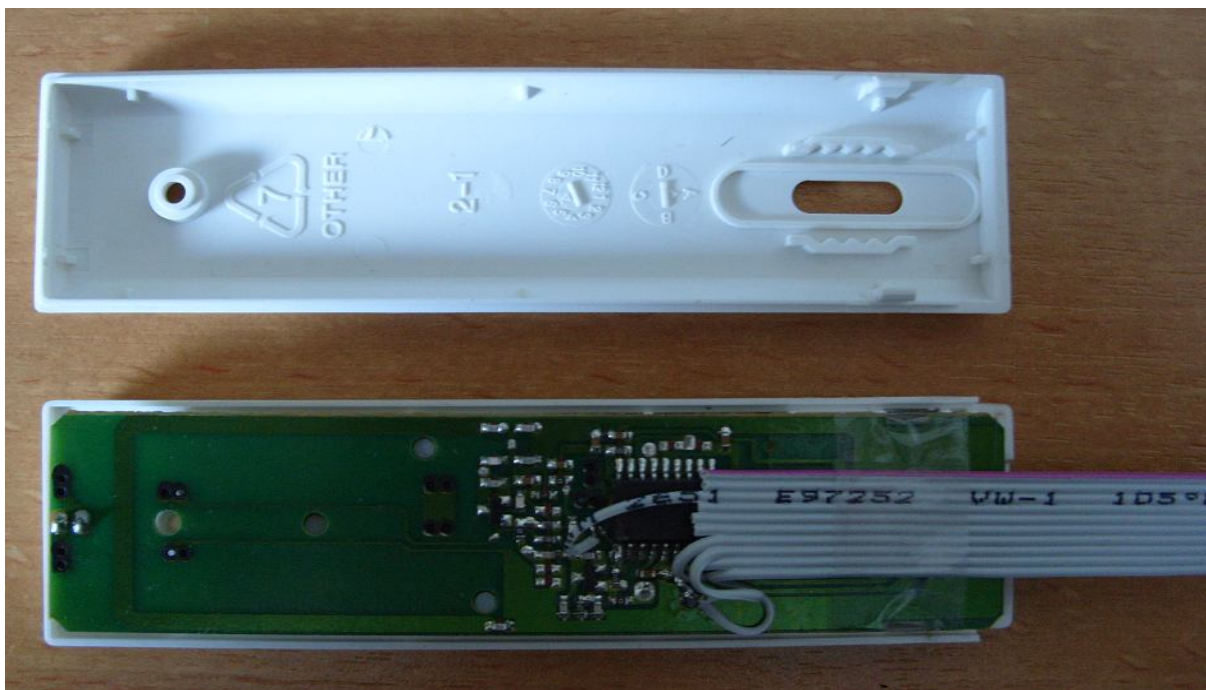
Pin 8 = GND, connect to the battery - pad

Pin 9 = not used

Pin 10 = GND, connect to the battery - pad



Secure the cable to the PCB using tape. Remove the slide switch from the case.



Put the case back together and you're done. Store the remaining parts in a safe place in case you want to restore the remote into it's original state.

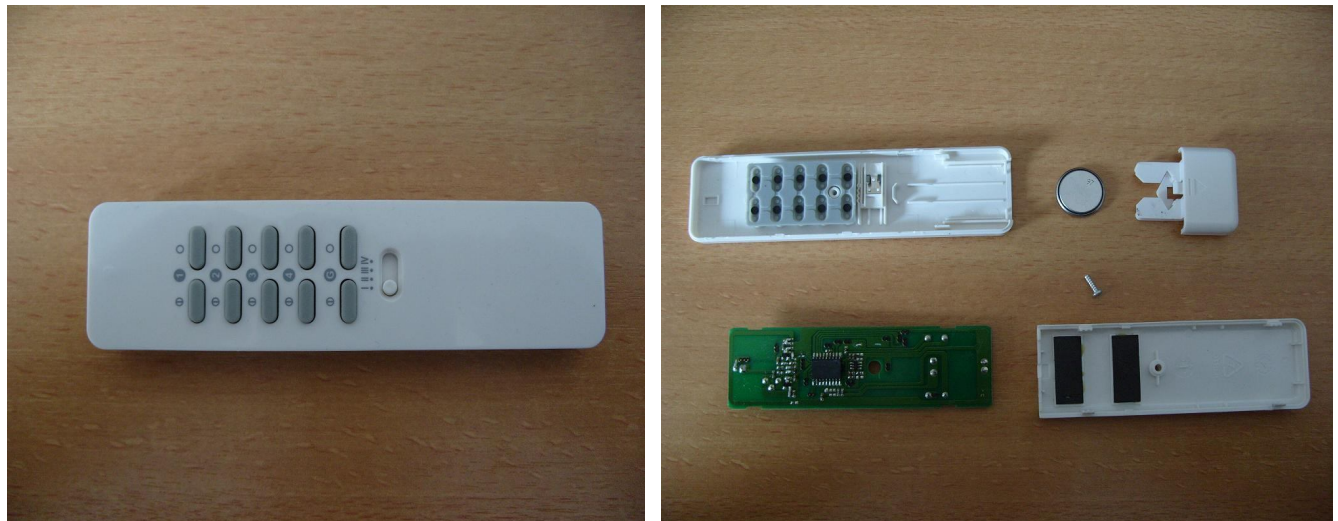


How to connect a KlikAanKlikUit AYCT-102 remote controll to your cassiopei.

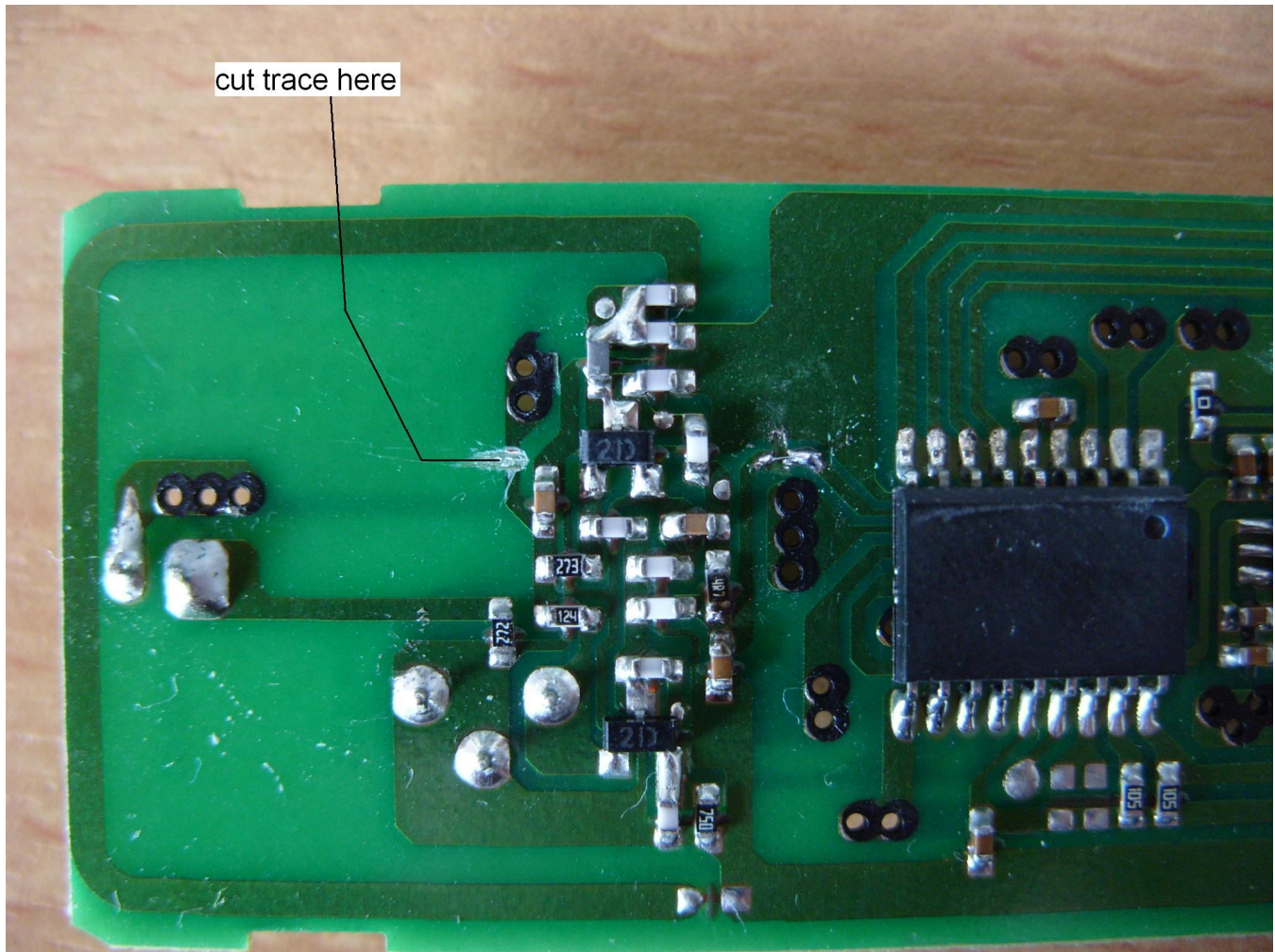
This is how the modified AYCT-102 remote control will look like when the mod is complete. As you can see, it plugs directly into your cassiopei's expansion connector. This modification no longer allows the use of the buttons, the Cassiopei (or to be more precise, the CBM computer will be in full control).

This is how the modified remote control will look like when the mod is complete. As you can see, it plugs directly into your cassiopei's expansion connector.

Remove the battery of the AYCT-102 remote control and using a Philips screwdriver open the remote control. Store the battery in a safe place, we won't need this any more. The cassiopei's power supply is sufficient to drive the remote control.



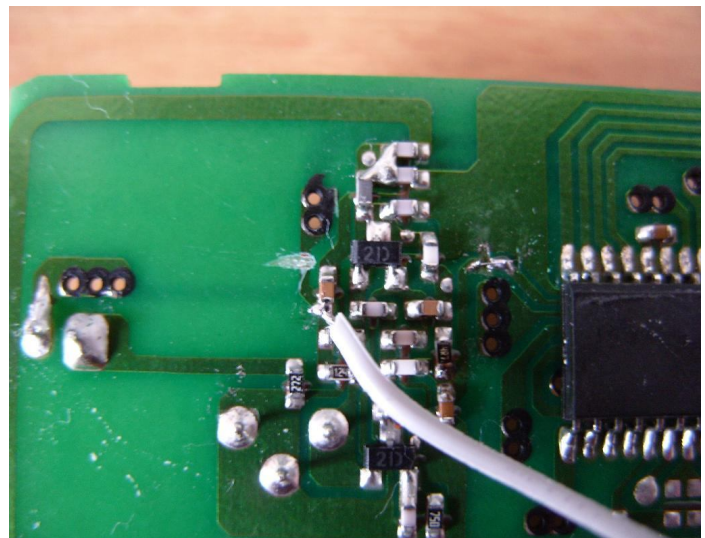
Using a sharp knife we cut the trace as indicated by the photo. This cut will disconnect PIC-1 of the remote control IC from the RF circuit. So that we can connect the RF circuit the IO pin of the expansion connector of the Cassiopei.



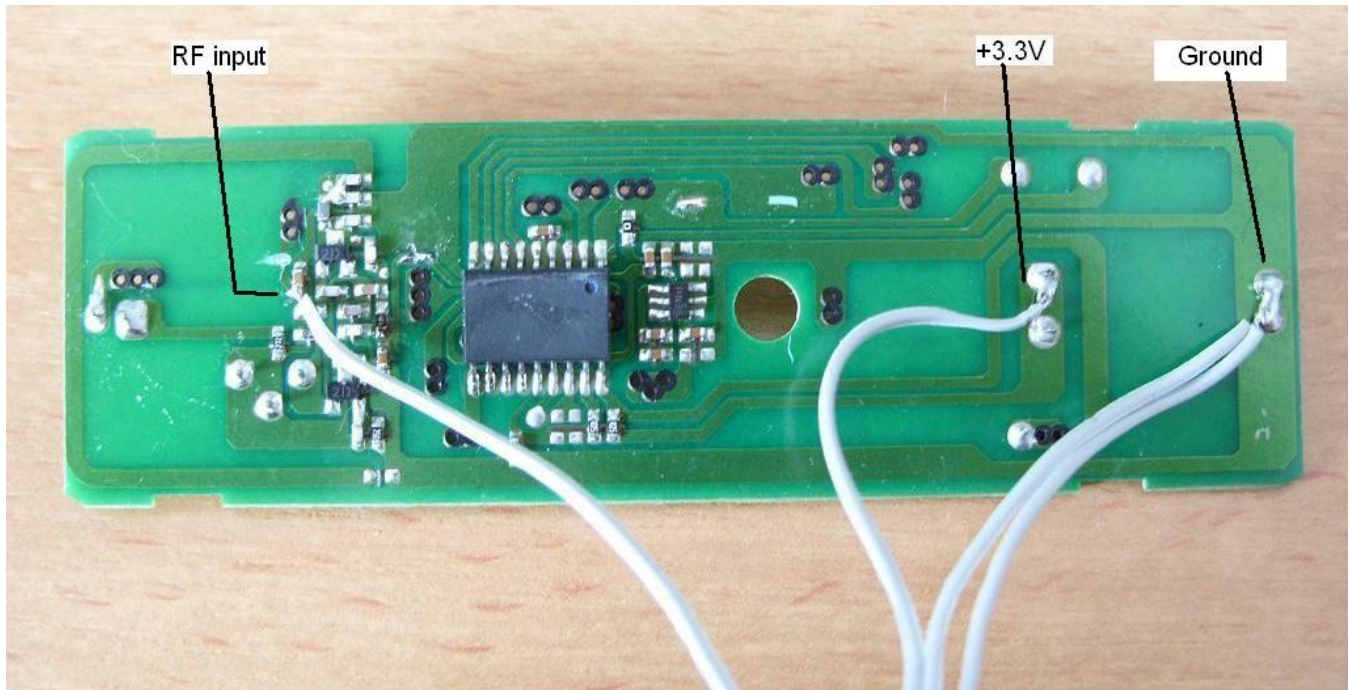
The picture below shows how the signal from the Cassiopei's expansion port should be connected to the RF circuit of the remote control.

10 pole ribbon cable

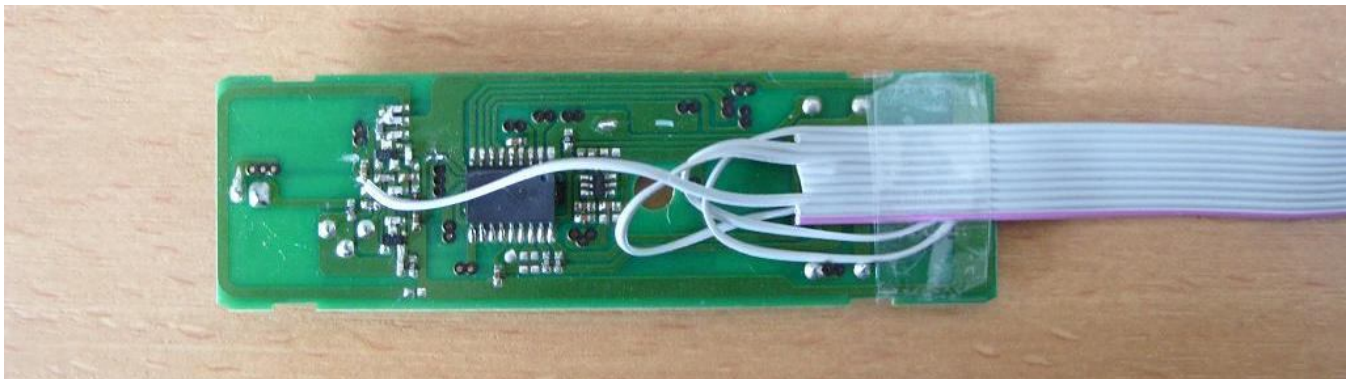
- Pin 1 = not used
- Pin 2 = not used
- Pin 3 = signal, connect to input of RF circuit
- Pin 4 = not used
- Pin 5 = not used
- Pin 6 = not used
- Pin 7 = +3.3V, connect to the battery + pad
- Pin 8 = GND, connect to the battery - pad
- Pin 9 = not used
- Pin 10 = GND, connect to the battery - pad



Also connect the other 3 wires, the 2 ground wires connect to the pad (-) of the battery connector. The +3.3V wire connects to the pad (+) of the battery connector.



Secure the cable to the PCB using tape. Remove the slide switch from the case.



Put the case back together and you're done. Store the remaining parts in a safe place in case you want to restore the remote into it's original state.

