THE ULTIMATE GUIDE

to the

NEO 6502

OLIMEX Ltd.
2 Pravda St., P.O. Box 237,
Plovdiv 4000 BULGARIA

Contact: Mr. Tsvetan Usunov
Email: info@olimex.com
Voice: +359-32-626259, +359-32-267407, +359-32-621270

Welcome – please read!

Welcome to the modern retro computer world, where you can experience the technology from the 70s and 80s, but with a modern spin on it!

This document covers both the Neo6502 and Neo6502pc computer. Detailed specifications and the differences between the two can be found in Appendix A.

> Neither of the devices (the Neo6502 and Neo6502pc) are turn-key solutions. Both devices require intermediate electronics and computer use knowledge. While both devices have appeared in social media as an out-of-the-box video game platform, it will require that you read this document, so that you gain the best experience!

Please Note

Regardless of the function you are hoping to utilize the Neo6502 or Neo6502pc, you must be familiar with the process of reprogramming (also known as flashing firmware) the 2MB flash memory utilized by the RP2040. The firmware defines what function the Neo6502 or Neo6502pc will perform. Current firmware available provide a BASIC interpreter (NeoBASIC) that is continues to be developed and improved, an Apple][emulator (using the real W6502), and an Oric Atmos. Many and other firmware packages are currently being developed, so explore the various user forums, Discord, and Facebook to discover the endless possibilities of the Neo6502 and Neo6502pc.

Please read the Programming the RP2040 Section (page 7)

Both devices require that you obtain or supply the following for proper operation:

Neo6502

- USB-C Power Source (5v, 0.2 amps, minimum to start w/o peripherals,).
- A USB cable with a USB-A on one end, and the appropriate end that will connect to your computer *(used to re-program the RP2040).*
- *Optional,* enclosing case for the Neo6502, *available from Olimex.*
- *Optional*, USB-A Flash Drive (*highly recommend USB3, ~8 GB*), formatted FAT32.
- *Optional*, USB Hub (*Olimex USB-NeoHub is highly recommended for compatibility*).
- *Optional,* USB Gamepad.

Neo6502pc

- USB-C Power Source (5v, 1 amp).
- A USB cable with a USB-C on one end, and the appropriate end that will connect to your computer.
- USB-A Flash Drive (highly recommend USB3, ~8 GB), formatted FAT32.
- USB Keyboard *(wired and wireless w/USB dongle), bluetooth is not supported.*
- *Optional,* USB Gamepad.

Document Formatting Conventions

In this documentation, the following standards will be used:

- Code examples and parameters will be displayed in a fixed-space font.
- Parameters are listed in italics, and have a descriptive name to know what should be substituted, and the entire parameter is replaced with the described value.
- Parameters enclosed with square braces [] are optional.
- The symbol ↵ indicates the entry of the key carriage return (return key).
- The use of an ellipse (…) indicates a range, starting the alphanumeric and ending with the last alphanumeric.

Other Symbols

	Shown near a paragraph or example code indicates an unusual feature to which you should pay attention to syntax details for proper execution.
	Shown near a paragraph or example code indicates the need to be careful or alert when using following the directions or code example. It may cause a crash or problem there you may lose your work and may need to restart the Neo6502.

Immediate -vs- Deferred Execution Commands

Many, but not all commands can be executed immediately by just typing the command at the beginning of a line and pressing return.

All commands (with a few exceptions) can be used within a program and can be preceded with a line number to add it to a program. Execute those commands as part of the program by typing "run ↵", and to see a listing of your program type “list ↵ “.

MOS -vs- mos

There are several locations in the guide where the acronym “mos” is used. To help ensure that the correct acronym expansion is interpreted correctly, here is a key:

- MOS (all uppercase letters) = Machine Operating System.
- mos (all lowercase letters) = metal oxide semiconductor.

Also note, that the command “mos” can be both “MOS” and “mos” and the command comes from the uppercase “MOS” (Machine Operating System).

Table of Contents

About the Neo6502

The Neo6502 is a standalone modern retro computer with a real W65C02 processor and RP2040 co-processor. This small device works 3-times-faster than any of the other recent 6502 competitors and 30-times-faster than 6502 based machines from the 1980s.

The "Neo" name was used two reasons: First it implies a modern design; Second came from the analogy with the movie The Matrix where the W65C02 lives in virtual world – thinking it has real memory, video and keyboard – however in reality it is all virtual and emulated with the RP2040.

Both the Neo6502 and Neo6502pc are open-source hardware (https://freedomdefined.org/OSHW), with all CAD files and firmware available to support the future development of software and enhancements to the hardware.

There are two models available:

- The Neo6502, an open circuit board computer (2 revisions, A & B).
- The Neo6502pc, a Neo6502 enclosed in a 3D-printed case with a LCD display, USB ports, UEXT and 6502 interface ports and more.

More technical specifications can be found in Appendix A (page 67). More information about the Neo6502 project, please refer to the Neo6502 website: http://www.neo6502.com

About the W65C02 processor

The W65C02, being a more modern 6502 than the old retro metal oxide semiconductor chip (mos) – in that it can go much faster than was possible in the 1970s and 1980s. The W65C02 can even be overclocked to 16 MHz, but on the Neo6502 it is running at 6.25 Mhz, which is closer to the clock speed of the Amiga and Atari ST than the Atari or C64, and a lot faster than when most of the retro games were being coded.

The Neo6502 features a real W65C02S processor, which does all the computing with real timing versus emulation, but the real power of the machine coms from the RP2040 which provides the memory, video, keyboard input, and additional IO for SPI, I2C, UART, and so on.

Things like complex math (multiplication, floating-point) and graphics are also handled by the RP2040, acting like a co-processor. Unlike other similar architectures, the RP2040 has direct memory access (providing the memory for the 6502) so there are no additional big data transfers between the chips to wait for, making things all much more efficient.

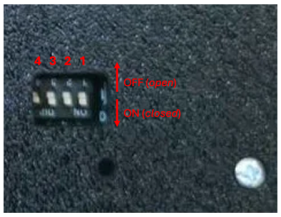
The processor gets 64kb of RAM, but there is 2 MB of flash memory on board, access to USB flash drive for storage via USB or expansion port (for SD card support), and there is a 40-pin connector that offers up a bus of all the 6502 signals and pins that can be used to interface with or use for experiments. The UEXT ports already support quite a few modules from Olimex that support UEXT specification (https://www.olimex.com/Products/Modules/).

Programming the RP2040

The process of programming the RP2040 is a fairly easy process, *however*, it has a very specific manner and steps that must be followed to have a successful reprogramming.

NOTES

- Some firmware images require all switches of the configuration switch block be in the on (closed) position.



Prerequisites

- Your computer should be on, and you must be logged in and have the desktop present. Best experience comes with no CPU intensive tasks running on your computer.

- You have the latest version of the firmware that you want to use downloaded to your computer. *It is highly recommended that you download the firmware file from the "source of truth" (the developer's Github repository or website).*

  A firmware file come in various sizes and names, based on the functionality it performs, however it will always have the uf2 file extension.

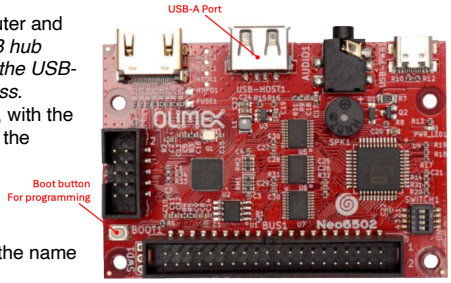
- Make sure the Neo6502 device has been powered down.

RP2040 programming for the Neo6502

Required hardware:

- A computer with a USB port and a modern operating system.
- A Neo6502 computer.
- A USB cable with a USB-A on one end, and the appropriate end that will connect to your computer.

Steps:



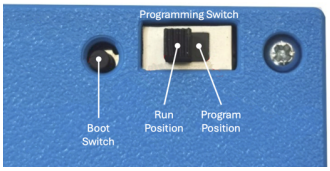
1. Connect the USB cable between your computer and the Neo6502 USB-A port. *If you have a USB hub connected or any other device connected to the USB-A port, please disconnect it during this process.*
2. Press and hold the "boot" button (bottom left, with the UEXT port on the left and the W6502 bus on the bottom). *Ensure you have heard or felt the button depress with a satisfying "click".*
3. Turn the power on.
4. Release the "boot" button.
5. A volume will appear on your computer with the name "RPI-RP2".
6. Copy the appropriate UF2 file to the "RPI-RP2" volume.
7. **Do not be alarmed**, as soon as the copy is finished, the volume will disappear. *This indicates that the firmware has been successfully uploaded and programming has begun and will only take a few seconds.*
8. Reconnect the USB hub and other devices that were removed on step 1.

RP2040 programming for the Neo6502pc

Required hardware:

- A computer with a USB port and a modern operating system.
- A Neo6502pc computer.
- A USB cable with a USB-C on one end, and the appropriate end that will connect to your computer.

Steps:



1. Connect the USB Cable between your computer and the Neo6502 USB-C port (with the LCD facing up, the USB-C port on the left).
2. Slide the programming switch on the back of the Neo6502pc to the programming position (with the switch facing up and in the upper left corner – move to the right-most position).
3. Press and hold the "boot" button (to the left of the programming switch). *Ensure you have heard or felt the button depress with a satisfying "click".*
4. Continue to press the "boot" button and turn the power on.
5. Release the "boot" button.
6. A volume will appear on your computer with the name "RPI-RP2".
7. Copy the appropriate UF2 file to the "RPI-RP2" volume.
8. **Do not be alarmed**, as soon as the copy is finished, the volume will disappear. *This indicates that the firmware has been successfully uploaded and programming has begun and will only take a few seconds.* The Neo6502pc will automatically reboot using the new firmware.
9. Move the programming switch back to "run" position.

SUCCESS

Based on the firmware that was just flashed, the Neo6502pc will now operate within the firmware function. Please refer to the documentation that comes with the firmware to know the next steps. The most popular firmware and their next steps are provided in this document.

Programming Troubleshooting

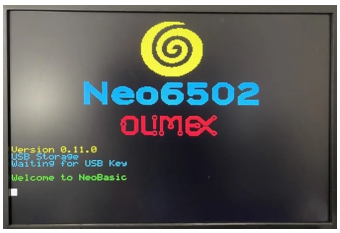
- If you are using a Neo6502pc, ensure the programming switch in in the "program" position.
-

Current Firmware

The following are accurate as of the August 4th, 2024 revision of this document.

NeoBasic (codename: Morpheus)

Maintained by Paul Robson (paul@robsons.org.uk)
GitHub Repository: https://github.com/paulscottrobson/neo6502-firmware
Obtain the firmware from the repository link



1. Within the Github respository, navigate to the releases section (right side)
2. Click on the link (release number). This will take you to the releases list.
3. Locate and click the zip file to download it.
4. Unzip the file.
5. Locate the "`firmware_usb.uf2`" file.
   The "`firmware_sd.uf2`" file is used when you are using the SDCard adapter.
6. Follow the directions above to program the RP2040 on page 6.

Please refer to the NeoBasic section (page 11) for more information.

Apple][and //e Emulation

Maintained by Veselin Sladkov
(veselin.sladkov@gmail.com)
Obtain the firmware from:
https://github.com/vsladkov/reload-emulator
The firmware source code is found on the repository; however, it is not compiled into a uf2 file. You can download the uf2 firmware file from Olimex's FTP site:
https://ftp.olimex.com/Neo6502/

1. Click the link to open the Olimex FTP site.
2. Click and download the "`blank_disk_for_apple2e_code_development_apple2e_ProDOS_2_4_3.zip`" file.
3. Unzip it, and copy the "`ProDOS_2_4_3.po`" to a flash drive.
4. Follow the directions above to program the RP2040 on page 6, with the "`apple2e.uf2`" file.

You can replace the "`ProDOS_2_4_3.po`" with other disk images that can be found on the internet. Check out the Apple][section on the Internet Archive (https://archive.org/details/softwarelibrary_apple_games) as well as other locations.

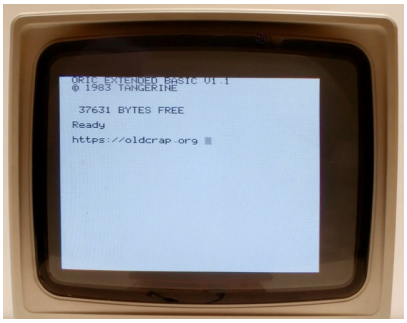
Apple][TotalReplay

Maintained by Veselin Sladkov (veselin.sladkov@gmail.com)
Obtain the firmware from: https://github.com/vsladkov/reload-emulator
The firmware source code is found on the repository; however, it is not compiled into a uf2 file. You can download the uf2 firmware file from Olimex's FTP site: https://ftp.olimex.com/Neo6502/

1. Click the link to open the Olimex FTP site.
2. Click and download two files:
   - "`Total Replay v5.1.hdv`" file.
   - "`apple2e-5.uf2`" file.
3. Copy the "`Total Replay v5.1.hdv`" to a flash drive.
4. Follow the directions above to program the RP2040 on page 6, with the "`apple2e-5.uf2`" firmware file.

If successful, turning on the Neo6502 device, will present you with the TotalReplay title screen. All games can be played with a keyboard and some games support the USB gamepad, your milage by vary.

Oric Atmos



Maintained by Veselin Sladkov (veselin.sladkov@gmail.com)
Obtain the firmware from:
https://github.com/vsladkov/reload-emulator
The firmware source code is found on the repository; however, it is not compiled into a uf2 file.

You can download the uf2 firmware file from Olimex's FTP site:

`https://ftp.olimex.com/Neo6502/uf2/oric_960x540_372MHz.uf2`

This is an older version, and no compiled version with updated firmware is available as a download, as you must have copies of the Oric ROMs. You will need to compile it yourself or ask folks on social media if an updated compiled version is available.

NeoBASIC

Written by Paul Robson

NeoBASIC runs very fast compared to most 6502-based home computers back in the day that were 2 Mhz or less. Compared to the FPGA-assisted Commander X16 that runs at 8 Mhz, it underperforms versus the Neo6502 in this BASIC line draw demo, partially due to the inefficient BASIC interpreter.

You can program right on the Neo6502 using NeoBASIC or use the emulators that will run on Windows, Linux, and MacOS desktop operating systems. Cross-development is possible even with NeoBASIC by using the Python script to tokenize (text to BASIC binary) or detokenize (BASIC binary to text) your basic listing ready for execution.

There are some NeoBASIC based games that have been written that really show off some of the very powerful features:

• **Galaxians** arcade game written by Paul Robson *Still a work-in-progress* • **Space Invaders** arcade game	• **Frogger** arcade game • written by Paul Robson *Still a work-in-progress*

Programming Reference and Technical Documentation

The information that follows describes the NeoBASIC interpreter. It is for experienced programmers and includes explanations of all BASIC statements and functions, memory tables, sprite usage, and file formats.

NeoBASIC Technical Reference

This version of BASIC was written with concepts brought over from more modern BASIC interpreters and other programming languages that make it more powerful and create a lot less spaghetti code† that BASIC programs generally tended to create.

Line Numbers

In NeoBASIC, line numbers are only used to provide an order to the program where declaration order is important. Line numbers should not be used as a reference within the program, and while GOSUB and GOTO are valid BASIC commands, their use is highly discouraged.

It is recommended that a main program exist with functions and procedures used along with proper use of “for”, “while” and “repeat” flow control programming syntax.

> **Note:** If line numbers are used and the use of GOTO and GOSUB are also used, a program could easily stop working properly in the event that the program is ever “renumbered”. A program can be renumbered either deliberately using the “renumber” command or as a result of editing a program with the built-in editor.

Procedure Placement

Procedure declarations should be at the end of a program, with a preceding “end” statement before the first “proc” statement. Should NeoBASIC encounter a proc statement during execution, the program will halt and present a syntax error.

```
10 print “Say Something”
20 print “----------------------”
25 a$=”Something to say!”
30 call saysomething(a$)
40 end
50 proc saysomething(a$)
60    print a$
70 endproc
```

Comments

A code should include comments. Good comments don't repeat the code or explain it. They clarify its intent. Comments should explain, at a higher level of abstraction than the code, what you're trying to do.

Comments can be added to code as a line by itself or added to the end of an existing line by using the single quote following by your comment. Comments should be limited to alphanumeric, and caution should be used when using quotes and other special characters.

Note, after entering a comment, the BASIC tokenizer will display it as a single quote followed by you comment within double quotes. A comment is basically an ignored string at execution.

† **Spaghetti code** is a pejorative phrase for difficult-to-maintain and unstructured computer source code. See: https://en.wikipedia.org/wiki/Spaghetti_code#

Binary Operators

Precedence	Operator	Notes
4	*	Multiplication operator
4	/	Forward slash is floating point divide. 22/7 is 3.142857
4	\	Backward slash is integer divide, 22\7 is 3
4	%	Modulus of integer division ignoring signs
4	>>	Logical shift right, highest bit zero
4	<<	Logical shift left
3	+	Addition operator
3	-	Subtraction operator
2	<	Less than
2	<=	Less than or equal
2	>	Great than
2	>=	Greater than or equal
2	<>	Not equal to
2	=	Equal to
1	&	Binary AND operator on integers
1	|	Binary OR operator on integers
1	^	Binary XOR operator on integers

In the above table, an expression using the above operators will evaluate as `-1` for `true`, and `0` for `false`.

Functions

Arithmetic and Boolean Functions

Function	Description (*function and return value)*
`atan(n)`	Calculate the arctangent (the inverse tangent function) of `n` in degrees
`atan2(y,x)`	Calculates the arctangent (the inverse tangent function) of `y,x` if `x` equals `0`, atan2 returns `π/2` if `y` is positive, `-π/2` if `y` is negative, or `0` if `y` is `0`.
`cos(n)`	Calculate the cosine of `n` (`n` must be in degrees)
`exp(n)`	Calculates the exponential value of a floating-point argument n (e^n, where `e` equals `2.17128128...`)
`FALSE`	Return constant `0`, a known falsey value. The value zero is always consider false.
`int(n)`	Return the whole part of the float value `n`. Integers are unchanged.
`log(n)`	Calculate the natural logarithm (e.g. ln2) of `n`.
`max(a,b)`	Return the largest of `a` and `b` (numbers or strings)
`min(a,b)`	Return the smallest of `a` and `b` (numbers or strings)
`pow(a,b)`	Returns `a` raised to the power `b`; the result is always floating point.
`rand(n)`	Returns a random integer, where `0 < x < n`. *The value returned will be between `0` and `n-1`.*
`rnd(0)`	Returns a random number where `0 < x < 1`. *The value (`zero`) passed is ignored.*
`sin(n)`	Calculate the sine of `n` (`n` must be in degrees)
`sqr(n)`	Calculates the square root of `n`
`tan(n)`	Calculates the tangent of `n` (`n` must be in degrees)
`TRUE`	Return constant `-1`, a known truthy value. *Any value greater than 0 or less than 0 is considered true.*

File System and I/O Functions

Function	Description (*function and return value*)
`eof(filename)`	Returns non-zero value if at end of file *`filename`*.
`exists(filename)`	Returns true (-1) if the file *`filename`* exists, false (0) otherwise.
`locale string`	Sets the locale to the ISO 3166-1 alpha-2 country code *`string`* e.g. locale "de" for Germany, “us” for United States of America. Country code list: https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2
`mos(command)`	Returns zero if the command was successful, and non-zero error if it was unsuccessful or generated an error. **See**: MOS Commands (page 33)

BASIC Interpreter Functions

Function	Description (*function and return value*)
`err`	Current error number
`erl`	Current error line number

String Functions

Function	Description (*function and return value)*
`asc(s$)`	Return ASCII value of first character or zero for empty string.
`chr$(n)`	Convert ASCII decimal value to a character.
`instr(str$,search$)`	Returns the first position of `search$` in `str$`, indexed from 1. Returns `zero` if not found.
`isval(s$)`	Converts string to number, returns `-1` if okay, `0` if fails.
`left$(a$,n)`	Left most number of characters (`n`) of `a$`.
`len(a$)`	Return length of string in characters.
`lower$(a$)`	Convert a string to lower case.
`mid$(a$,f[,s])`	Characters from `a$` starting at `f` (1 indexed), `s` characters, `s` is optional and defaults to the rest of the line.
`right$(a$,n)`	Rightmost n characters of `a$`.
`spc(n)`	Returns a string containing the number (`n`) spaces.
`str$(n)`	Convert a number (`n`) to a string.
`upper$(a$)`	Convert a string to upper case.
`val(s$)`	Convert string to number, error if bad number.

Hardware Information Functions

Function	Description (*function and return value*)
`alloc(n)`	Allocate `n` bytes of memory, return address
`analog(n)`	Read voltage level on pin `n` -- returns a value from `0` to `4095`
`deek(a)`	Read word value at `a`
`event(v,r)`	event takes an integer variable and a fire rate (`r`) in `1/100` seconds, and uses the integer variable to return `-1` at that rate. If the value in `v` is zero, it resets (if you pause say), if the value in `v` is `-1` the timer will not fire -- to unfreeze, set it to `zero` and it will resynchronize.
`havemouse()`	Return non-zero if a mouse is connected.
`Himem`	First byte after end of memory -- the stack is allocated below here, and string memory below that.
`inkey$()`	Return the key stroke if one is in the keyboard buffer, otherwise returns a `n` empty string.
`idevice(device)`	Returns true if i2c device present.
`iread(device,register)`	Read byte from I2C Device Register
`joycount()`	Read the number of attached joypads, not including keyboard emulation of one.
`joypad([index],dx,dy)`	Reads the current joypad. The return value has `bit 0` set if A is pressed, `bit 1` set if B is pressed. Values `-1,0` or `1` are placed into `dx,dy` representing movement on the D-Pad. If there is no gamepad plugged in (*at the time of writing it doesn't work*) the key equivalents are WASDOP and the cursor keys. If `[index]` is provided it is a specific joypad (from `1,0` is the keyboard), otherwise it is a composite of all of them.

Hardware Information Functions (*continued)*

Function	Description (*function and return value)*
`key(n)`	Return the state of the given key. The key is the USB HID key scan code `n`.
`mouse(x,y[,scroll])`	Reads the mouse. The return value indicates button state (`bit 0 left, bit 1 right`), and the mouse position and the scrolling wheel position are updated into the given variables.
`notes(c)`	Return the number of notes outstanding on channel `c` including the one currently playing -- so will be zero when the channel goes silent.
`page`	Return the address of the program base (e.g. the variable table)
`peek(a)`	Read byte value at `a`
`pin(n)`	Return value on UEXT pin `n` if input, output latch value if output.
`point(x,y)`	Read the screen pixel at coordinates `x,y`. This is graphics data only.
`spoint(x,y)`	Reads the color index on the sprite layer. `0` is transparency
`tab(n)`	Advance to screen column `n` if not past it already.
`time()`	Return time since power on in `1/100` seconds.
`uhasdata()`	Return true if there is data in the UART Receive buffer.
`vblanks()`	Return the number of vertical-blanks since power on. This is updated at the start of the vertical-blank period.

Commands

Flow Control Commands

Control Structure	Description

`do`
`  ...`
`exit`
`  ...`
`loop`
	Provides an infinite loop structure, with the ability to use the `exit` command to break out of the loop at any point.
	
	`10 index = 0`
	`20 do`
	`30  print "Hello World!"`
	`40  index = index + 1`
	`50  if index > 10 then exit`
	`60 loop`

`end`	End the current running program

`for var = start`
`to/downto end`
`...`
`next var`
	Provides a controlled loop (for / next) using a range of values.
	Note this is non-standard for/loop control, as there are limitations.
	Limitations:
	• The index must be an integer.
	• The step is controlled used "to" (1) or "downto" (-1).
	
	`next` cannot specify an index and cannot be used to terminate loops (*using the wrong index).* Execution must operate in order and flow cannot be stopped arbitrarily. *The variable after next is ignored.*
	
	`10 for count = 10 downto 1`
	`20  print count`
	`30 next count`
	`40 print "Blast off!"`

`gload filename`	Load filename into graphics memory.
`gosub expr` †	Call subroutine at line number. A `return` command will return control to the line after the gosub call.
`goto expr`†	Transfer execution to a line number.
`return`†	*See `gosub`.*

† Provided to use only for porting code to the NeoBASIC. Continued use is not recommended as line numbers can change. The `renumber` command or use of the built-in editor will change line numbers and not update `gosub` and `goto` calls.

Flow Control Commands (*continued…)*

Control Structure	Description
`if` *expr* `then` *nn*	Provide a single conditional, where execution is transferred to line number `nn` if the {expr} is true. **NOTE:** `if {expr} goto` *nn* does not work.

`if {expr}:`
`..`
`else`
`..`
`endif`
	Provides an `if-then-else` condition. More than one line can be used between `if` and `endif` or between `if` and `else` and `else` and `endif`. *The `else` clause is optional.*
	
	Code between `if` and `else` will only execute if the `{expr}` evaluates as true, and the code between `else` and `endif` will only execute if the `{expr}` evaluates as false.

`on error` *nn*	Install an error handler, when execution is transferred to line number nn when an error occurs (effectively a `goto` command).

`repeat`
`..`
`until` *expr*
	Provides a finite loop structure that will end when the expression following the `until` evaluates as true, else it will repeat while the expressing following the `until` evaluates as false.

`run`	Will execute the program in memory at the first and lowest line number.
`stop`	Will terminate the program with an error.
`wait` *s*	Waits for `{s}`, where `{s}`is 1/100 seconds.

`while` *expr*
`...`
`wend`
	Provides a finite loop structure that will end when the expression following the `while` evaluates as false, else it will repeat while the expressing following the `while` evaluates as true.

File System and I/O Commands

Command	Description
`close [handle]`	Close a file by handle. The handle is optional, and if not provided, all files will be closed.
`input #{channel},{var},{var}`	Reads a sequence of variables from the open file.
`ireceive d,a,s`	Receive bytes starting at a, count s to or from device d.
`itransmit d,a,s`	Send bytes starting at a, count s to or from device d.
`isend device,data`	Send data to i2c {device}; this is comma separated data, numbers or strings. If a semicolon is used as a separator e.g. 4137; then the constant is sent as a 16-bit value.
`iwrite dev,reg,b`	Write byte to I2C Device Register
`load "file[",address]`	Load file to BASIC space or given address. The last quote is optional if the parameter `address` *is not used.*
`mos "command"`	Execute Machine Operating System (MOS) command within the quotes. Commands include `cat`, `cd`, `md`, `copy`, `del`, and more. **See**: MOS Commands (page 33)

File System and I/O Commands (*continued…)*

Command	Description
`open input|output [channel][,file]`	Open a file for input or output on the given channel, using the given file name. Output erases the current file. This gives an error if the file does not exist; rather than trap this error it is recommended to use the `exists()` function if you think the file may not be present.
`print [string | var]`	This will output the contents that following the command at the current cursor position. A string is encapsulated in double-quotes. A var can be a string or number variable. You can concatenate strings and vars together with the + symbol. If you end a print statement with a semi-colon, then the next print will follow at the end of this print, effectively appending to the output. Use `str$(var)` to concatenate an integer with the string.
`print #{channel},{expr},{expr}`	Writes a sequence of expressions to the open file.
`print line #channel.var.var`	Prints a line to an output channel as an ASCII file, in LF format (e.g. lines are separated by character code 10). This can be mixed with the above format, *but* the sequence has to be the same; you can't write a string using print line and read it back with input and vice versa. All variables must be strings.
`run "program"`	Load & Run program. *The last quotation mark is optional.*
`save "file[",adr,sz]`	Save BASIC program or memory from `adr` length `sz`. The last quote is option if the `adr` *or* `sz` *parameters are not used.*
`sreceive a,s`	Receive bytes starting at a, count s to SPI device
`stransmit a,s`	Send bytes starting at a, count s to SPI device
`ssend data`	Send data to SPI device; this is comma separated data, numbers or strings. If a semicolon is used as a separator e.g. 4137; then the constant is sent as a 16-bit value.
`ureceive d,a,s`	Receive bytes to/from the UART starting at a, count s
`utransmit d,a,s`	Send bytes to/from the UART starting at a, count s
`usend device, data`	Send data to UART; this is comma separated data, numbers or strings. If a semicolon is used

Commented [GJ1]: Finish this entry

BASIC Commands

Command	Description
`' {string}`	Comment. This is a string for syntactic consistency. If you type in the comment without the speech marks, which is easier, the speech marks will be added automatically. See comments (page 12).
`assert expr[,msg]`	Error generated if {expr} is zero, with optional message.
`cls`	Clear the graphics screen to current background color. This does not clear sprites.
`cursor x,y`	Set the text cursor position to position x,y on the screen.
`data const,const,…`	DATA statement. Strings must be enclosed in quote marks. *A read statement will read the data found in the data statements. See:* `read` and `restore`.
`defchr ch,....`	Define UDG ch (192-255) as a 6x7 font -- should be followed by 7 values from 0-63 representing the bit pattern of the graphic, so if these numbers are converted to a 6 digit binary numbera '1' represents a pixel that is 'on', and a '0' represents a pixel that is off.
`delete`	Delete a line or range of lines
`dim array(n,[m]), $...`	Dimension a one or two dimension string or number array, up to 255 items.
`edit`	Basic Screen Editor
`fkey`	Lists the defined function keys
`fkey key,string`	Define the behavior of F1..F10 -- the characters in the string
`ink fgr[,bgr]`	Set the ink foreground and optionally background for the console.
`input var`	Will wait for input that ends when hitting return. The content entered will be stored in the variable `var`.
`let var=expr`	Assignment statement. The `let` is optional.

Commented [GJ2]: Finish this entry

BASIC Commands *(continued…)*

<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td>library [from][,][to]
library</td><td>The library command allows you to hide a section of code from the program listing. It can be performed multiple times with different ranges to hide discontinuous sections of code.

Entering the command without any parameters will unhide all sections of code and renumber the program starting at 1000.

The hidden “library” code persists in files saved, and will remain hidden even when the program is loaded. 0</td></tr>
<tr><td>list</td><td>List an entire program</td></tr>
<tr><td>list [from]</td><td>List a program starting at line number from</td></tr>
<tr><td>list [from][,][to]</td><td>List a program starting at line number from to the line number to.</td></tr>
<tr><td>list procedure()</td><td>List the lines associated with a particular procedure. Must have the () following the name of the procedure.</td></tr>
<tr><td>local var,var</td><td>Define local variables within a procedure scope. Variables with the same name as local variables will be restored and then conclusion of the procedure execution, and local values will be lost if not assigned to global variables or passed out via the “def” parameter modifier.</td></tr>
<tr><td>mouse cursor n</td><td>Select mouse cursor n
<table>
<tr><td>0</td><td>White Arrow (default)</td><td>20</td><td>White arrow</td></tr>
<tr><td>1</td><td>Fly</td><td>21</td><td>Black arrow</td></tr>
<tr><td>2</td><td>Unknown</td><td>22</td><td>Small arrow</td></tr>
<tr><td>3</td><td>Target</td><td>23</td><td>Very small arrow</td></tr>
<tr><td>4</td><td>Unknown</td><td>24</td><td>Very small pointer</td></tr>
<tr><td>5</td><td>Unknown</td><td>25</td><td>Arrow</td></tr>
<tr><td>6</td><td>Unknown</td><td>26</td><td>Finger pointing</td></tr>
<tr><td>7</td><td>Unknown</td><td>27</td><td>Watermelon</td></tr>
<tr><td>8</td><td>Dartboard</td><td>28</td><td>Lime</td></tr>
<tr><td>9</td><td>Magnifying glass</td><td>29</td><td>Lemon</td></tr>
<tr><td>10</td><td>Hourglass</td><td>30</td><td>Dinosaur</td></tr>
<tr><td>11</td><td>Paint bucket (fill)</td><td>31</td><td>Crown</td></tr>
<tr><td>12</td><td>Right-angle tool</td><td>32</td><td>Dice</td></tr>
<tr><td>13</td><td>Right-angle tool</td><td>33</td><td>Globe</td></tr>
<tr><td>14</td><td>Unknown</td><td>34</td><td>Christmas tree</td></tr>
<tr><td>15</td><td>Pencil</td><td>35</td><td>Olympic rings</td></tr>
<tr><td>16</td><td>Paintbrush</td><td>36</td><td>Mountain peek</td></tr>
<tr><td>17</td><td>Unknown</td><td>37</td><td>Flower</td></tr>
<tr><td>18</td><td>Yin-yang</td><td rowspan="2">38</td><td rowspan="2">Floppy disk</td></tr>
<tr><td>19</td><td>Paint brush</td></tr>
</table>

The above list was accurate as of firmware version 0.99.1, and could change.</td></tr>
</table>

BASIC Commands *(continued…)*

Command	Description
`mouse show`	Show the mouse on the screen
`mouse TO x,y`	Position mouse cursor
`new`	Erase any program in memory.
`old`	Reverses the "`new`" statement. *Please note, that this could fail, depending on what actions or changes to memory has taken place since "`new`" was executed.*
`palette c,r,g,b`	Set color `c` to `r,g,b` values. Values are all 0 – 255, however it is actually 3:2:3 color, so the result will be approximations. An example would be `palette 7,255,128,0` which sets color 7 (normally white) to R = 255, G = 128, B = 0 which is orange.
`palette clear`	Reset palette to default

`proc name([p1,p2,…])`
`…`
`endproc`
	Creates a procedure, that can optionally have parameters. If parameters are used, when the procedure is called, the parameters must match (number and types) exactly.
	
	Parameters can be defined as reference parameters and will return values.
	`proc name( [ref] p1, [ref] p2,[ref]...)`
	Any change to the variables will be passed back at the end of the procedure in the same variables that were in the parameter section.
	
	Parameters cannot be arrays.
	
	NOTE: Procedures should be defined at the end of a basic program and an "end" statement need to precede the declaration section. If the BASIC program execution hits a proc declaration, a syntax error will be presented.

`call name ([p1,p2,…])`	Call named procedure with optional parameters. If the procedure is defined with reference variables, then values are returned at the end of the procedure call.

Commented [GJ3]: Need an example

Command	Description
`renumber [start]`	Renumber the program in memory starting at 1000, or from the optional parameter *start*. The `renumber` command will not change any line numbers used with `goto` or `gosub`. *These are commands are not recommended for use and should be used at your own risk.*
`read var, …`	Read variables from data statements. Variable type must match the data (string or integer) in the data statements.
`restore`	Restore data pointer to the beginning of the `data` statements. Performing a read will read from the very first data constant.
`restore line`	Restore data pointer to line number *line*. The `restore` command uses line numbers, which are not guaranteed to remain the same in a program. *These are commands are not recommended for use and should be used at your own risk.*
`tilemap addr,x,y`	Define a `tilemap`. The `tilemap` data format is in the API. The `tilemap` is stored in memory at `addr`, and the offset into the

Commented [GJ4]: What does this do?

Interfacing with hardware

Command	Description
`clear [address]`	Clear out stack, strings, reset all variables. If an address is provided, then memory above that will not be touched by BASIC. Note because this resets the stack, it cannot be done in a loop, subroutine or procedure -- they will be forgotten. Also clears the sprites and the sprite layer.
`doke addr,data`	Write word to address
`mon`	Enter the machine code monitor
`pin pin,value`	Set UEXT {pin} to given value.
`pin pin INPUT`	output
`poke addr,data`	Write byte to address
`sys address`	Call 65C02 machine code at given address. Passes contents of variables A,X,Y in those registers.
`uconfig baud[,prt]`	Set the baud rate and protocol for the UART. Currently only 8N1 is supported.

Graphics Commands

Command	Description
`from x,y`	Sets the origin position, can be repeated and optional.
`to x,y`	Draw the element at x,y or between the current position and x,y depending on the command. So you could have text "Hello" to 10,10 or rect 0,0 to 100,50
`by x,y`	Same as to but x and y are an offset from the current position
`x,y`	Set the current position without doing the action
`ink c`	Modifier to a graphic command to change what color c the command will use.
`ink a,x`	Modify the color that is used in the graphics commands by using the screen color and performing a binary AND with parameter a, and performing a binary OR with parameter x.
`solid`	Fill in rectangles and ellipses. For images and text, forces black background.
`text {str} to x,y`	Draw/place text at t c0dc0he specified x,y coordinates. `text "Hello World!" to 20,20`
`rect`	`rect {solid | frame} x1,y1 to x2,y2` Will draw a rectangle with the upper-left-corner (`x1,y1`) to the bottom-right-corner (`x2,y2`). **Note:** Once used, rect and ellipse commands will continue using the solid or frame state until changed. **See** `frame, solid`

Commented [GJ5]: So can frame and solid be applied to more than rect and ellipse?

Command	Description
`ellipse`	`ellipse {solid | frame} x1,y1 to x2,y2`
`frame`	A modifier for rectangles and ellipses, where it will only draw the outline, not filling it in. `rect frame x1,y1 to x2,y2` will draw an empty rectangle.
`solid`	A modifier for rectangles and ellipses, where it will only draw a filled in rectangle or ellipse. `rect solid x1,y1 to x2,y2` will draw a filled in rectangle.
`dim n`	Set the scaling to *n* (for `text`, `image`, and `tilemap` only), and must be an integer. `text "Hello" dim 2 to 10,10 to 10,100` will draw the word “Hello” at double its size! Tiles can only be scaled at 1 or 2 (when scaling at 2, tiles are drawn at a size of 32x32, versus the scale 1 of 16 x16).
`move`	
`plot`	
`line`	
`flip`	flip is an optional modifier for the sprite command. 0 = no flip; 1= horizontal flip; 2 = vertical flip; or 3 = both vertical and horizontal.
`anchor`	
`image`	
`sprite`	

Pixel Colors

These colors are approximations and will vary depending on the type and image adjustments of the display.

Pixel	Hex	Color			Pixel	Hex	Color		
0	$80	Black *Transparent*	#000000		8	$88	Black	#000000	
1	$81	Red	#ff0044		9	$89	Dark Grey	#555544	
2	$82	Green	#00ee33		10	$8A	Dark Green	#008855	
3	$83	Yellow	#ffee22		11	$8B	Orange	#ffaa00	
4	$84	Blue	#112255		12	$8C	Dark Orange	#aa5533	
5	$85	Magenta	#772255		13	$8D	Brown	#887799	
6	$86	Cyan	#22aaff		14	$8E	Pink	#ffccaa	
7	$87	White	#ffffee		15	$8F	Light Grey	#cccccc	

Sprite Commands

Sprite commands closely resemble the graphics commands.

They begin with `SPRITE n` which sets the working sprite.

Options include:

- `IMAGE n` which sets the image.
- `TO x,y` which sets the position
- `FLIP n` which sets the orientation.
- `ANCHOR n` which sets the anchor point
- `BY x,y` which sets the position by offset.

~~With respect to the latter, this is the position from the TO and is used to do attached sprites e.g. you might write.~~

Example:

```
SPRITE 1 IMAGE 2 TO 200,200 SPRITE 2 IMAGE 3 BY 10,10
```

Which will draw Sprite 1 and 200,200 and sprite 2 offset at 210,210. It does not offset a sprite from its current position.

As with graphics commands not all options are required, as they are options, which applies modifiers to the command. You can simply use `SPRITE 1 IMAGE 3`.

`SPRITE` can also take the single command `CLEAR`; this resets all sprites and removes them from the display.

> **Note:** Sprite 127 is used to display the turtle sprite when turtle graphics are used. If turtle graphics are not going to be used, then this sprite can be overwritten and be used as any other sprite.

Implementation notes

- Up to 128 sprites are supported. Sprites are drawn by the RP2040 processor ~~and is not hardware~~, so keep in mind that more sprites could mean slower system performance.

Additionally, the sprites are currently done with XOR drawing, which causes effects when they overlap. This should not be relied on (it may be replaced by a clear/invalidate system at some point), but the actual implementation should not change.
This is an initial sprite implementation and is quite limited.
(The plan is to add a feature like the animation languages on STOS and AMOS which effectively run a background script on a sprite)
Sprite Support
=spritex(n) =spritey(n)
These return the x and y coordinates of the sprites draw position (currently the centre) respectively.
= hit(sprite#1,sprite#2,distance)
The hit function is designed to do sprite collision. It returns true if the pixel distance between the centre of sprite 1 and the centre of sprite 2 is less than or equal to the distance.
So if you wanted to move a sprite until it collided with another sprite, assuming both are 32x32, the collision distance would be 32 (the distance from the centre to the edge of both sprites added together), so you could write something like :
x = 0
repeat

```
x = x + 1: sprite 1 to x,40
until hit(1,2,32)
```

In my experience of this the distance needs to be checked experimentally, as it affects the 'feel' of the game ; sometimes you want near exact collision, sometimes it's about getting the correct feel. It also depends on the shape and size of the sprites, and how they move.
I think it's better than a simple box collision test, and more practical than a pixel based collision test which is very processor heavy.

MOS Commands

Using the MOS commands to access OS disk/file functionality.

MOS commands can be used in four different ways:

- On the NeoBASIC command line, prefixing the command with an asterisk.
  Example: `*del myfile.txt`

- On the NeoBASIC command line, use the mos BASIC command. Surround the MOS command in quotes.
  Example: `mos "del myfile.txt"`

- Within a NeoBASIC program, you can use the mos BASIC function. Surround the MOS command in quotes.
  Example: `if mos("del myfile.txt") > 0 then …`

Available MOS Commands

`cat`	Prints out the file listing of the current directory.	`ren`	Will rename a file or directory. Works identically as the copy command.
`del`	Delete the specified file. Can be used to delete a directory, only if it is empty. *If a directory is not empty, the command generates an error.*	`md`	Make a directory. Specify the name of the directory you want to create.
`copy`	Copy on path to another. Requires two parameters, the first is the source path and file, and the second is the destination path and file. If no path is given, but different names are given, then you get a copy of the source file specified in the first parameter in the current directory. You cannot make copies of directories.	`cd`	Change Directory. Specify the name of the directory that you wish to traverse.
`file`	Will verify that a file (*not a directory*) exists. Return zero if it exists, and non-zero if it does not exist.		

Standard Unix POSIX paths are used with the MOS commands.

`./`	Current directory
`../`	Go up in the hierarchy, relative to the current directory (*sometime referred as the going back a directory*).
`/`	The root or top-level directory.

The Inline Assembler

The inline assembler works in a very similar way to that of the BBC Micro, except that it does not use the square brackets [and] to delimit assembler code. Assembler code is in normal BASIC programs.

A simple example shown below (in the samples directory). It prints a row of 10 asterisks.

Most standard 65C02 syntax is supported, except currently you cannot use lsr a ; it has to be just lsr (and similarly for rol, asl, ror,inc and dec).

You can also pass A X Y as variables. So you could delete line 150 and run it with X = 12: sys start which would print 12 asterisks.

Line	Code	Notes
100	`mem = alloc(32)`	Allocate 32 bytes of memory to store the program code.
110	`for i = 0 to 1`	We pass through the code twice because of forward referenced labels. This actually doesn't apply here.
120	`p = mem`	P is the code pointer -- it is like $* = {xx} - it means put the code here
130	`o = i * 3`	Bit 0 is the pass (0 or 1) Bit 1 should display the code generated on pass 2 only, this is stored in 'O' for options.
140	`.start`	Superfluous -- creates a label 'start' -- which contains the address here
150	`ldx #10`	Use X to count the starts
160	`.loop1`	Loop position. We can't use loop because it's a keyword
170	`lda #42`	ASCII code for asterisk
180	`jsr $fff1`	Monitor instruction to print a character
190	`dex`	Classic 6502 loop
200	`bne loop1`	
210	`rts`	Return to caller
220	`next`	Do it twice and complete both passes
230	`sys mem`	BASIC instruction to 'call 6502 code'. Could do sys start here.

[ ] Operator

The `[]` operator is used like an array, but it is a syntactic equivalent of `deek` and `doke`, e.g. reading and writing 16 bytes. `mem[x]` means the 16 bit value in `mem + x $* 2`, so if `mem = 813` then `mem[2] = -1` writes a `16-bit` word to `817` and `818`, and print mem[2] reads it. The index can only be from `0 ..127` The purpose of this is to provide a clean readable interface to data in 65C02 and other programs running under assembly language; often accessing elements in the 'array' as a structure.

Zero-Page Usage

Neo6502 is a clean machine, rather like the Sharp machines in the 1980s. When BASIC is not running it has no effect on anything, nor does the firmware. For example, unlike on the Commodore 64, changing some zero-page locations can cause crashes.

However, BASIC does make use of zero-page. At the time of writing this is memory locations `$10-$41`. These can however be used in machine code programs called via `SYS`. Only 4 bytes of that usage is system critical (the line pointer and the stack pointer), those are saved on the stack by `SYS`, so even if you overwrite them it does not matter.

However, you can't use this range to store intermediate values *between* sys calls. It is advised that you work usage backwards from `$FF` (as BASIC is developed forwards from `$10`). It is very unlikely that these will meet in the middle.

`$00` and `$01` are used on BASIC boot (and maybe other languages later) but this should not affect anything.

Raspberry PI 2040 Messaging API

The Neo6502 uses a Raspberry PI 2040 to provide memory, graphics support, file I/O capabilities and external (UEXT) interface. While the BASIC and other programming languages implement many of these capabilities as command and functions, many of the routines can be access natively via the RP2040 messaging API.

The Neo6502 API is a messaging system. There are no methods to access the hardware directly. Messages are passed via the block of memory from $FF00 to $FF0F, as specified in the "API Messaging Addresses" table below.

API Messaging Addresses

Address	Type	Notes
`$FF00`	Group	Group selector and status. Writing a non-zero value to this location triggers the routine specified in $FF01. The system will respond by setting the 'Error', 'Information', and 'Parameters' values appropriately. Upon completion, this memory location will be will cleared.
`$FF01`	Function	A command or function within the selected Group. For example, Group 1 Function 0 writes a value to the console; and Group 1 Function 1 reads the keyboard.
`$FF02`	Error	Return any error values, 0 = no error.
`$FF03:7`	Status	Set (1) if the ESCape key has been pressed. This is not automatically reset.
Note: FF03:6 through FF03:0 are not used.		

`$FF04`
…
`$FF0B`
	Parameters	This memory block is notated in this document as Params[0] through Params[7], or as a composite list or range (eg: Params[1,2], Params[0..7]).
		
		Some functions require parameters in these locations and some return values in these locations, and other function do neither.

The above address map and the tables describing the functions found in this section can be found in numerous sections of the firmware release download:

- `examples/assembly/neo6502.inc`
- `examples/C/neo6502.h`

API Commands/Functions are grouped by functionality.

Command/Function Parameters are notated in this document as Params[0] through Params[7], or as a list or range (eg: Params[1,2], Params[0..7]).

Note: That these are referring to a mapping to memory locations. The numbers represent offsets from the Parameters base address $FF04. Ie: the actual bytes are not necessarily all distinct "parameters" in the conventional sense. Depending on the routine, a logical parameter may be an individual byte, one or more bits of a byte interpreted as a composite or bit-field, or multiple adjacent bytes interpreted as 16 or 32 bit values.

For example: The list Params[0,1] would indicate a single logical parameter, comprised of the two adjacent bytes $FF04 and $FF05. The range Params[4..7] would indicate a single logical parameter, spanning consecutive bytes between $FF08 and $FF0B.

Note: **Strings referenced by Parameters are not ASCIIZ, but are length-prefixed.** The first byte represents the length of the string (not counting itself). The string begins at the second byte. Consequently, strings must be 255 bytes or less (not counting the length header).

Using RP2040 messaging API in NeoBASIC

`proc send.message(g,f)`	Define a procedure to “send a message” using parameters f (function), and p (parameters)
`  while peek($FF00):wend`	Wait until the messaging API is ready to accept a message (all previous commands in queue have completed)
`  poke $FF01,f:poke $FF00,g`	Poke the function and group. If there are parameters to pass, you must have those entered into memory (poke) prior to performing “poke $ff00,g”, as this call the function immediately.
`  while peek($FF00):wend`	Wait until for the function to complete.
`endproc`	End the procedure.

System

G/F	Function	Description and Example
`1,0`	DSP Reset	Resets the messaging system and component systems. Normally, should not be used.
`1,1`	Timer	Deposit the value (32-bits) of the 100Hz system timer into Parameters:0..3.
`1,2`	Key Status	Deposits the state of the specified keyboard key into Parameter:0. State of keyboard modifiers (Shift/Ctrl/Alt/Meta) is returned in Parameter:1. The key which to query is specified in Parameter:0.
`1,3`	Basic	Loads and allows the execution of BASIC via an indirect jump through address zero.
`1,4`	Credits	Print the Neo6502 project contributors (stored in flash memory).
`1,5`	Serial Status	Check the serial port to see if there is a data transmission.
`1,6`	Locale	Set the locale code specified in Parameters:0,1 as upper-case ASCII letters. Parameter:0 takes the first letter and Parameter:1 takes the second letter. For example: French (FR) would require Parametr 0 being $46 and Parameter 1 being $52
`1,7`	System Reset	System Reset. This is a full hardware reset. It resets the RP2040 using the Watchdog timer, and this also resets the 65C02.
`1,8`	MOS	Perform a MOS command.
`1,10`	Write character to debug	Writes a single character to the debug port (the UART on the Pico, or stderr on the emulator). This allows maximum flexibility.
`1,11`	Return Version Information	Reads the current version: `major.minor.patch` into parameters: 0-2. These values are guaranteed to be in the range 0 – 255.

Commented [GJ6]: Make sure to expand this to include WIKI and "/" command.

Commented [JG7]: Where are these values found?

Commented [GJ8R7]: Should they be in the API shortcut variable list?

Commented [GJ9]: Where do you put the command to run? What parameters?

Console

G/F	Function	Description and Example
`2,0`	Write Character	Console out. *This is a duplicate of* function `2,6` *for backward compatibility.*
`2,1`	Read Character	Read and remove a key press from the keyboard queue into Parameter:0. This is the ASCII value of the keystroke. If there are no key presses in the queue, Parameter:0 will be zero. **Note:** This function is better suited for text input, but not for games. See function `7,1` (read default controller) is better suited for games, as this only detects key presses. It does not include the ability to check whether the key is currently down or not.
`2,2`	Console Status	Check to see if the keyboard queue is empty. If it is, Parameter:0 will be $FF, otherwise it will be $00
`2,3`	Read Line	Input the current line below the cursor into Parameters:0,1 as a length-prefixed string; and move the cursor to the line below. Handles multiple-line input.
`2,4`	Define Hotkey	Define the function key F1..F10 specified in Parameter:0 as 1..10 to emit the length-prefixed string stored at the memory location specified in Parameters:2,3. F11 and F12 cannot currently be defined.
`2,5`	Define Character	Define a font character specified in Parameter:0 within the range of 192..255. Fill bits 0..5 (columns) of Parameters:1..7 (rows) with the character bitmap.
`2,6`	Write Character	Write the character specified in Parameter:0 to the console at the cursor position. Refer to Section "Console Codes" for details.
`2,7`	Set Cursor Pos	Move the cursor to the screen character cell Parameter:0 (X), Parameter:1 (Y).
`2,8`	List Hotkeys	Display the current function key definitions
`2,9`	Screen Size	Returns the console size in characters, in Parameter:0 (height) and Parameter:1 (width).
`2,10`	Insert Line	This is an internal function (inserting a blank line on the console) and is not officially supported. *It is not recommended for use and maybe obsoleted or the functionality may change.*
`2,11`	Delete Line	This is an internal function (deletes line from the console) and is not officially supported. *It is not recommended for use and maybe obsoleted or the functionality may change.*

Commented [GJ10]: Look up, provide links?

2,12	Clear Screen	Clears the screen. Equivalent to the “cls” BASIC command.
2,13	Get Cursor Position	Returns the current screen character cell of the cursor in Parameter:0 (X), Parameter:1 (Y).
2,14	Clear Text Region	Erase all characters within the rectangular region specified in Parameters:0,1 (begin X,Y) and Parameters:2,3 (end X,Y).
2,15	Set Text Color	Sets the foreground color to Parameter:0 and the background color to Parameter:1
2,16	Cursor Inverse	This is an internal function (inverts/swaps the foreground and background colors – normal -vs- inverse) and is not officially supported. *It is not recommended for use and maybe obsoleted or the functionality may change.*
2,17	Tab	Moves the cursor to the right until it reaches the position in Parameter 0. This is an internal helper function. *It is not recommended for use and maybe obsoleted or the functionality may change.*
2,18	Read foreground and background colors	Read the foreground and background RGB colors into Param[0] and Param[1]
2,19	Show/Hide Cursor Reversing	Set the cursor visibility to Param[0]. This is reset by clearing the screen.

Commented [JG11]: Why does this mention paper? Ink?

File I/O

G/F	Function	Description and Example
3,1	List Directory	Display the file listing of the present directory.
3,2	Load File	Load a file by name into memory. On input: Parameters:0,1 points to the length-prefixed filename string; Parameters:2,3 contains the location to write the data to. If the address is $FFFF, the file will instead be loaded into the graphics working memory, used for sprites, tiles, images. On output: Error location contains an error/status code.
3,3	Store File	Saves data in memory to a file. On input: Parameters:0,1 points to the length-prefixed filename string; Parameters:2,3 contains the location to read data from; Parameters:4,5 specified the number of bytes to store. On output: Error location contains an error/status code.
3,4	File Open	Opens a file into a specific channel. On input: Parameter:0 contains the file channel to open; Parameters:1,2 points to the length-prefixed filename string; Parameter:3 contains the open mode. See below. Valid open modes are: 0 opens the file for read-only access; 1 opens the file for write-only access; 2 opens the file for read-write access; 3 creates the file if it doesn't already exist, truncates it if it does, and opens the file for read-write access. Modes 0 to 2 will fail if the file does not already exist. If the channel is already open, the call fails. Opening the same file more than once on different channels has undefined behaviour, and is not recommended.
3,5	File Close	Closes a particular channel. On input: Parameter:0 contains the file channel to close. If this is $FF this closes all open files.
3,6	File Seek	Seeks the file opened on a particular channel to a location. On input: Parameter:0 contains the file channel to operate on; Parameters:1..4 contains the file location. You can seek beyond the end of a file to extend the file. However, whether the file size changes when the seek happens, or when you perform the write is undefined behavior.
3,7	File Tell	Returns the current seek location for the file opened on a particular channel. On input: Parameter:0 contains the file channel to operate on. On output: Parameters:1..4 contains the seek location within the file.
3,8	File Read	Reads data from an opened file. On input: Parameter:0 contains the file channel to operate on. Parameters:1,2 points to the destination in memory,

		or $FFFF to read into graphics memory. Parameters:3,4 contains the amount of data to read. On output: Parameters:3,4 is updated to contain the amount of data actually read. Data is read from the current seek position, which is advanced after the read.
3,9	File Write	Writes data to an opened file. On input: Parameter:0 contains the file channel to operate on; Parameters:1,2 points to the data in memory; Parameters:3,4 contains the amount of data to write. On output: Parameters:3,4 is updated to contain the amount of data actually written. Data is written to the current seek position, which is advanced after the write.
3,10	File Size	Returns the current size of an opened file. On input: Parameter:0 contains the file channel to operate on. On output: Parameters:1..4 contains the size of the file. This call should be used on open files, and takes into account any buffered data which has not yet been written to disk. Consequently, this may return a different size than Function 3,16 "File Stat".
3,11	File Set Size	Extends or truncates an opened file to a particular size. On input: Parameter:0 contains the file channel to operate on; Parameters:1..4 contains the new size of the file.
3,12	File Rename	Renames a file. On input: Parameters:0,1 points to the length-prefixed string for the old name; Parameters:2,3 points to the length-prefixed string for the new name. Files may be renamed across directories.
3,13	File Delete	Deletes a file or directory. On input: Parameters:0,1 points to the length-prefixed filename string. Deleting a file which is open has undefined behavior. Directories may only be deleted if they are empty.
3,14	Create Directory	Creates a new directory. On input: Parameters:0,1 points to the length-prefixed filename string.
3,15	Change Directory	Changes the current working directory. On input: Parameters:0,1 points to the length-prefixed path string.
3,16	File Stat	Retrieves information about a file by name. On input: Parameters:0,1 points to the length-prefixed filename string. Parameters:0..3 contains the length of the file; Parameter:4 contains the attributes bit-field of the file. If the file is open for writing, this may not return the correct size due to buffered data not having been flushed to disk. File attributes are a bitfield as follows: 0,0,0,Hidden, Read Only, Archive, System, Directory.
3,17	Open Directory	Opens a directory for enumeration. On input: Parameters:0,1 points to the length-prefixed filename string.

		Only one directory at a time may be opened. If a directory is already open when this call is made, it is automatically closed. However, an open directory may make it impossible to delete the directory; so closing the directory after use is good practice.
3,18	Read Directory	Reads an item from the currently open directory. On input: Parameters:0,1 points to a length-prefixed buffer for returning the filename. Parameters:0,1 is unchanged, but the buffer is updated to contain the length-prefixed filename (without any leading path); Parameters:2..5 contains the length of the file; Parameter:6 contains the file attributes, as described by Function 3,16 "File Stat". If there are no more items to read, this call fails and an error is flagged.
3,19	Close Directory	Closes any directory opened previously by Function 3,17 "Open Directory".
3,20	Copy File	Copies a file. On input: Parameters:0,1 points to the length-prefixed old filename; Parameters:2,3 points to the length-prefixed new filename. Only single files may be copied, not directories.
3,21	Set File Attributes	Sets the attributes for a file. On input: Parameters:0,1 points to the length-prefixed filename; Parameter:2 is the attribute bitfield. (See Stat File for details.) The directory bit cannot be changed. Obviously.
3,22	Check End of File (EOF)	Returns the end of file status of an opened file. On input: Parameter:0 contains the file channel to operate on. On output: Parameter:0 is non-zero if the file is at the end of the file. This call should be used on open files and may return an error if the file is closed.
3,32	List Filtered	Prints a filtered file listing of the current directory to the console. On input: Parameters:0,1 points to the filename search string. Files will only be shown if the name contains the search string (ie: a substring match).

Mathematics

G/F	Function	Description and Example
4,0	Addition	Register1 := Register 1 + Register2
4,1	Subtraction	Register1 := Register 1 - Register2
4,2	Multiplication	Register1 := Register 1 * Register2
4,3	Decimal Division	Register1 := Register 1 / Register2 (floating point)
4,4	Integer Division	Register1 := Register 1 / Register2 (integer result)
4,5	Integer Modulus	Register1 := Register 1 mod Register2
4,6	Compare	Parameter:0 := Register 1 compare Register2 : returns $FF, 0, 1 for less equal and greater.
4,7	Power	Register1 := Register 1 to the power of Register2 (floating point result whatever)
4,8	Distance (counter-rectangle)	Register1 := Square root of (Register1 * Register1) + (Register2 * Register2)
4,9	Angle calculation (arctangent2)	Register1 := arctangent2(Register 1,Register 2) - angle in degrees/radians
4,16	Negate	Register1 := -Register 1
4,17	Floor	Register1 := floor(Register 1)
4,18	Square Root	Register1 := square root(Register 1)
4,19	Sine	Register1 := sine(Register 1) angles in degrees/radians
4,20	Cosine	Register1 := cosine(Register 1) angles in degrees/radians
4,21	Tangent	Register1 := tangent(Register 1) angles in degrees/radians
4,22	Arctangent	Register1 := arctangent(Register 1) angles in degrees/radians
4,23	Exponent	Register1 := e to the power of Register 1
4,24	Logarithm	Register1 := log(Register 1) natural logarithm
4,25	Absolute Value	Register1 := absolute value(Register 1)
4,26	Sign	Register1 := sign(Register 1), returns -1 0 or 1
4,27	Random Decimal	Register1 := random float from 0-1
4,28	Random Integer	Register1 := random integer from 0 to (Register 1-1)
4,32	Number to Decimal	Helper function for tokenizer, do not use.
4,33	String to Number	Convert the length prefixed string at Parameters:4,5 to a constant in Register1.

4,34	Number to String	Convert the constant in Register1 to a length prefixed string which is stored at Parameters:4,5
4,35	Set Degree/Radian Mode	Sets the use of degrees (the default) when non zero, radians when zero.

Graphics

G/F	Function	Description and Example
5,1	Set Defaults.	Configure the global graphics system settings. Not all parameters are relevant for all graphics commands; but all parameters will be set by this command. So mind their values. Refer to Section "Graphics Settings" for details. The parameters are And, Or, Fill Flag, Extent, and Flip. Bit 0 of flip sets the horizontal flip, Bit 1 sets the vertical flip.
5,2	Draw Line	Draw a line between the screen coordinates specified in Parameters:0,1,Parameters:2,3 (begin X,Y) and Parameters:4,5,Parameters:6,7 (end X,Y).
5,3	Draw Rectangle	Draw a rectangle spanning the screen coordinates specified in Parameters:0,1,Parameters:2,3 (corner X,Y) and Parameters:4,5,Parameters:6,7 (opposite corner X,Y).
5,4	Draw Ellipse	Draw an ellipse spanning the screen coordinates specified in Parameters:0,1,Parameters:2,3 (corner X,Y) and Parameters:4,5,Parameters:6,7 (opposite corner X,Y).
5,5	Draw Pixel	Draw a single pixel at the screen coordinates specified in Parameters:0,1,Parameters:2,3 (X,Y).
5,6	Draw Text	Draw the length-prefixed string of text stored at the memory location specified in Parameters:4,5 at the screen character cell specified in Parameters:0,1,Parameters:2,3 (X,Y).
5,7	Draw Image	Draw the image with image ID in Parameter:4 at the screen coordinates Parameters:0,1,Parameters:2,3 (X,Y). The extent and flip settings influence this command.
5,8	Draw Tilemap	Draw the current tilemap at the screen coordinates specified in Parameters:0,1,Parameters:2,3 (top-left X,Y) and Parameters:4,5,Parameters:6,7 (bottom-right X,Y) using current graphics settings.
5,32	Set Palette	Set the palette colour at the index spcified in Parameter:0 to the values in Parameter:1,Parameter:2,Parameter:3 (RGB).
5,33	Read Pixel	Read a single pixel at the screen coordinates specified in Parameters:0,1,Parameters:2,3 (X,Y). When the routine completes, the result will be in Parameter:0. If sprites are in use, this will be the background only (0..15), if sprites are not in use it may return (0..255)
5,34	Reset Palette	Reset the palette to the defaults.

5,35	Set Tilemap	Set the current tilemap. Parameters:0,1 is the memory address of the tilemap, and Parameters:2,3,Parameters:4,5 (X,Y) specifies the offset into the tilemap, in units of pixels, of the top-left pixel of the tile.
5,36	Read Sprite Pixel	Read Pixel from the sprite layer at the screen coordinates specified in Parameters:0,1,Parameters:2,3 (X,Y). When the routine completes, the result will be in Parameter:0. Refer to Section "Pixel Colors" for details.
5,37	Frame Count	Deposit into Parameters:0..3, the number of v-blanks (full screen redraws) which have occurred since power-on. This is updated at the start of each v-blank period.
5,38	Get Palette	Get the palette colour at the index spcified in Parameter:0. Values are returned in Parameter:1,Parameter:2,Parameter:3 (RGB).
5,39	Write Pixel	Write Pixel index Parameter:4 to the screen coordinate specified in Parameters:0,1,Parameters:2,3 (X,Y).
5,64	Set Color	Set Color. Sets the current drawing colour to Parameter:0
5,65	Set Solid Flag	Set Solid Flag. Sets the solid flag to Parameter:0, which indicates either solid fill (for shapes) or solid background (for images and fonts)
5,66	Set Draw Size	Set Draw Size. Sets the drawing scale for images and fonts to Parameter:0
5,67	Set Flip Bits	Set Flip Bits. Sets the flip bits for drawing images. Bit 0 set causes a horizontal flip, bit 1 set causes a vertical flip.

Sprites

G/F	Function	Description and Example
6,1	Sprite Reset	Reset the sprite system.
6,2	Sprite Set	Set or update the sprite specified in Parameter:0. The parameters are : Sprite Number, X Low, X High, Y Low, Y High, Image, Flip and Anchor and Flags. Bit 0 of flags specifies 32 bit sprites. Values that are $80 or $8080 are not updated.
6,3	Sprite Hide.	Hide the sprite specified in Parameter:0.
6,4	Sprite Collision.	Parameter:0 is non-zero if the distance is less than or equal to Parameter:2 between the center of the sprite with index specified in Parameter:0 and the center of the sprite with index specified in Parameter:1.
6,5	Sprite Position	Deposit into Parameters:1..4, the screen coordinates of the sprite with the index specified in Parameter:0.

Controller

G/F	Function	Description and Example
7,1	Read Default Controller	This reads the status of the base controller into Parameter:0, and is a compatibility API call. The base controller is the keyboard keys (these are WASD+OPKL or Arrow Keys+ZXCV) or the gamepad controller buttons. Either works. The 8 bits of the returned byte are the following buttons, most significant first : Y X B A Down Up Right Left
7,2	Read Controller Count	This returns the number of game controllers plugged in to the USB System into Parameter:0. This does not include the keyboard based controller, only physical controller hardware.
7,3	Read Controller	This returns a specific controller status. Controller 0 is the keyboard controller, Controllers 1 upwards are those physical USB devices.

Sound

G/F	Function	Description and Example
8,1	Reset Sound	Reset the sound system. This empties all channel queues and silences all channels immediately.
8,2	Reset Channel	Reset the sound channel specified in Parameter:0.
8,3	Beep	Play the startup beep immediately.
8,4	Queue Sound	Queue a sound. Refer to Section #\ref{sound} "Sound" for details. The parameters are : Channel, Frequency Low, Frequency High, Duration Low, Duration High, Slide Low, Slide High and Source.
8,5	Play Sound	Play the sound effect specified in Parameter:1 on the channel specified in Parameter:0 immediately, clearing the channel queue.
8,6	Sound Status	Deposit in Parameter:0 the number of notes outstanding before silence in the queue of the channel specified in Parameter:0, including the current playing sound, if any.
8,7	Queue Sound Extended	Queue a sound. Refer to Section #\ref{sound} "Sound" for details. This is an extension of call 4 to support different waveform types and volumes. The source parameter is no longer used. The parameters are : Channel, Frequency Low, Frequency High, Duration Low, Duration High, Slide Low, Slide High, Sound Type and Sound Volume. All these are 16 bit parameters except the sound type and volume, and the channel number.
8,8	Get Channel Count	This returns the number of channels in Parameter #0

Turtle Graphics

G/F	Function	Description and Example
9,1	Turtle Initialize	Initialize the turtle graphics system. Parameter:0 is the sprite number to use for the turtle, as the turtle graphics system "adopts" one of the sprites. The icon is not currently re-definable, and initially the turtle is hidden.

9,2	Turtle Turn	Turn the turtle right by Parameter:0,1 degrees. Show if hidden. To turn left, turn by a negative amount.
		
		`API_TURTLE_LEFT  = #$010E; API parameter (turn -90 degrees)`
		`API_TURTLE_RIGHT = #$005A; API parameter (turn +90 degrees)`
		`API_TURTLE_FLIP  = #$00B4; API parameter (turn 180 degrees)`

9,3	Turtle Move	Move the turtle forward by Parameter:0,1 degrees, drawing in colour Parameter:2 if Parameter:3 is non-zero.
9,4	Turtle Hide	Hide the turtle.
9,5	Turtle Home	Move the turtle to the home position (in the center, pointing upward).
9,6	Turtle Show	Show the turtle.

UEXT port I/O

G/F	Function	Description and Example
10,1	UExt Initialize	Initialise the UExt I/O system. This resets the IO system to its default state, where all UEXT pins are I/O pins, inputs and enabled.
10,2	Write GPIO	This copies the value Parameter:1 to the output latch for UEXT pin Parameter:0. This will only display on the output pin if it is enabled, and its direction is set to "Output" direction.
10,3	Read GPIO	If the pin is set to "Input" direction, reads the level on pin on UEXT port Parameter:0. If it is set to "Output" direction, reads the output latch for pin on UEXT port Parameter:0. If the read is successful, the result will be in Parameter:0.
10,4	Set Port Direction	Set the port direction for UEXT Port Parameter:0 to the value in Parameter:1. This can be $01 (Input), $02 (Output), or $03 (Analogue Input).
10,5	Write I2C	Write to I2C Device Parameter:0, Register Parameter:1, value Parameter:2. No error is flagged if the device is not present.
10,6	Read I2C	Read from I2C Device Parameter:0, Register Parameter:1. If the read is successful, the result will be in Parameter:0. If the device is not present, this will flag an error. Use FUNCTION 10,2 first, to check for its presence.
10,7	Read Analog	Read the analogue value on UEXT Pin Parameter:0. This has to be set to analog type to work. Returns a value from 0..4095 stored in Parameters:0,1, which represents an input value of 0 to 3.3 volts.
10,8	I2C Status	Try to read from I2C Device Parameter:0. If present, then Parameter:0 will contain a non-zero value.
10,9	Read I2C Block	Try to read a block of memory from I2C Device Parameter:0 into memory at Parameters:1,2, length Parameters:3,4.
10,10	Write I2C Block	Try to write a block of memory to I2C Device Parameter:0 from memory at Parameters:1,2, length Parameters:3,4.
10,11	Read SPI Block	Try to read a block of memory from SPI Device into memory at Parameters:1,2, length Parameters:3,4.
10,12	Write SPI Block	Try to write a block of memory to SPI Device from memory at Parameters:1,2, length Parameters:3,4.

10,13	Read UART Block	Try to read a block of memory from UART into memory at Parameters:1,2, length Parameters:3,4. This can fail with a timeout.
10,14	Write UART Block	Try to write a block of memory to UART from memory at Parameters:1,2, length Parameters:3,4.
10,15	Set UART Speed and Protocol	Set the Baud Rate and Serial Protocol for the UART interface. The baud rate is in Parameters:0..3 and the protocol number is Parameter:4. Currently only 8N1 is supported, this is protocol 0.
10,16	Write byte to UART	Write byte Parameter:0 to the UART
10,17	Read byte from UART	Read a byte from the UART. It is returned in Parameter:0
10,18	Check if Byte Available	See if a byte is available in the UART input buffer. If available Parameter:0 is non zero.

Mouse

G/F	Function	Description and Example
11,1	Move display cursor	Positions the display cursor at Parameters:0,1,Parameters:2,3
11,2	Set mouse display cursor on/off.	Shows or hides the mouse cursor depending on the Parameter:0
11,3	Get mouse state	Returns the mouse position (screen pixel, unsigned) in x Parameters:0,1 and y Parameters:2,3, button state in Parameter:4 (button 1 is 0x1, button 2 0x2 etc., set when pressed), scroll wheel state in Parameter:5 as uint8 which changes according to scrolls.
11,4	Test mouse present	Returns non zero if a mouse is plugged in in Parameter:0
11,5	Select mouse Cursor	Select a mouse cursor in Parameter:0 ; returns error status if the cursor is not available.

Blitter

<table>
<tr><th>G/F</th><th>Function</th><th>Description and Example</th></tr>
<tr><td>12,1</td><td>Blitter Busy</td><td>Returns a non zero value in Parameter:0 if the blitter/DMA system is currently transferring data, used to check availability and transfer completion.</td></tr>
<tr><td>12,2</td><td>Simple Blit Copy</td><td>Copy Parameters:6,7 bytes of internal memory from Parameter:0:Parameters:1,2 to Parameter:3:Parameters:4,5.
Sets error flag if the transfer is not possible (e.g. illegal write addresses).
The upper 8 bits of the address are:
6502 RAM (00)
Video RAM (80,81)
Graphics RAM (90)</td></tr>
<tr><td>12,3</td><td>Complex Blit Copy</td><td>Copy a source rectangular area to a destination rectangular area. It's oriented toward copying graphics data, but can be used as a more general-purpose memory mover. The source and target areas may be different formats, and the copy will convert the data on the fly. For example, you can expand 4bpp source graphics (two pixels per byte) into the 1 pixel per byte framebuffer. However, the blitting is byte-oriented. So the source width is always rounded down to the nearest full byte.

Parameter (0) is the blit action:<table><tr><td>0</td><td>Copy</td></tr><tr><td>1</td><td>Copy masked - copy, but only where src is not the transparent value.</td></tr><tr><td>2</td><td>Solid masked - set target to constant solid value, but only where src is not the transparent value.</td></tr></table>See below for transparent/solid values.

Parameters (1,2) address of the source rectangle data.
Parameters (3,4) address of the target rectangle data.
The source and target rectangle data is laid out in memory as follows:<table><tr><td>0-2</td><td>24 bit address to copy from/to (address is address:page:0)</td></tr><tr><td>3</td><td>pad byte (must be zero)</td></tr><tr><td>4-5</td><td>Stride, in bytes. This is the value to add to the address to get from one line to the next</td></tr></table>Used for both source and target.

For example:
if blitting to the screen, a stride of screen width (320) would get to the next line.
A zero source stride would repeat a single line for the whole copy.
A negative target stride would draw from the bottom upward.
6 data format</td></tr>
</table>

		0 bytes. Supported for both source and target. 1 pairs of 4-bit values (nibbles). Source only. 2 8 single-bit values. Source only. 3 high nibble. Target only. 4 low nibble. Target only. 7 A constant to use as the "transparent" value for BLTACT_MASK and BLTACT_SOLID. Source only. Not used in target. 8 A constant to use as the "solid" value for BLTACT_SOLID. Source only. Not used in target. 9 Height. The number of lines to copy. Source only. Not used in target. The copy is driven by the source height. 10 - 11 Width. The number of values to copy for each line. Source only. Not used in target. The copy is driven by the source width.
12,4	Blit Image	Blits an image from memory onto the screen. The image will be clipped, so it's safe to blit partly (or fully) offscreen-images. **Parameter** / **Description** 0 The blit action (see function 3, Complex Blit) 1, 2 Address of the source rectangle data. 3, 4 X pixel coordinate on screen (signed 16 bit) 5, 6 Y pixel coordinate on screen (signed 16 bit) 7 Destination format, determines how framebuffer will be written: 0 write to whole byte. 1 unsupported 2 unsupported 3 write to high nibble only. 4 write to low nibble only. **NOTE:** Clipping operates at byte resolution on the source data. So, for example, if you blit a 1-bit image (format 2) to an x-position of -2, then the whole first byte will be skipped leaving 6 empty pixels on the left. Same happens on the right - either the whole source byte is used, or it'll be skipped.

Editor

G/F	Function	Description and Example
13,1	Initialize Editor	Initializes the editor
13,2	Reenter the Editor	Re-enters the system editor. Returns the function required for call out, the editors sort of 'call backs' - see editor specification.

```
; Convenience macros for Neo6502 applications programming
; SPDX-License-Identifier: CC0-1.0


;-----------------------------------------------------;
; Neo6502 Kernel jump vectors (see kernel/kernel.asm) ;
;-----------------------------------------------------;

ReadLine       = $FFEB
ReadCharacter  = $FFEE
WriteCharacter = $FFF1
WaitMessage    = $FFF4
SendMessage    = $FFF7


;--------------------------------------;
; Neo6502 Kernel API control addresses ;
;--------------------------------------;

ControlPort = $FF00
API_COMMAND    = ControlPort + 0 ; function group address
API_FUNCTION   = ControlPort + 1 ; function address
API_ERROR      = ControlPort + 2 ; function error codes
API_STATUS     = ControlPort + 3 ; misc hardware status codes (bit-field)
API_PARAMETERS = ControlPort + 4 ; function parameters base address (+0-7)


;-----------------------------------------------;
; Neo6502 Kernel API control codes (see api.pdf) ;
;-----------------------------------------------;

; Status Information
API_ERROR_NONE = #$00 ; error code
API_STATUS_ESC = #$07 ; flag

; System functions (Group 1)
API_GROUP_SYSTEM     = #$01 ; API function group
API_FN_TIMER         = #$01 ; API function
API_FN_KEY_STATUS    = #$02 ; API function
API_FN_BASIC         = #$03 ; API function
API_FN_CREDITS       = #$04 ; API function
API_FN_SERIAL_STATUS = #$05 ; API function
API_FN_LOCALE        = #$06 ; API function
API_FN_RESET         = #$07 ; API function
```

```
; Console functions (Group 2)
API_GROUP_CONSOLE     = #$02 ; API function group
API_FN_READ_CHAR      = #$01 ; API function
API_FN_CONSOLE_STATUS = #$02 ; API function
API_FN_READ_LINE      = #$03 ; API function
API_FN_DEFINE_HOTKEY  = #$04 ; API function
API_FN_DEFINE_CHAR    = #$05 ; API function
API_FN_WRITE_CHAR     = #$06 ; API function
API_FN_SET_CURSOR_POS = #$07 ; API function
API_FN_LIST_HOTKEYS   = #$08 ; API function
API_FN_SCREEN_SIZE    = #$09 ; API function
API_FN_INSERT_LINE    = #$0A ; API function
API_FN_DELETE_LINE    = #$0B ; API function
API_FN_CLEAR_SCREEN   = #$0C ; API function
API_FN_CURSOR_POS     = #$0D ; API function
API_FN_CLEAR_REGIION  = #$0E ; API function
API_FN_SET_TEXT_COLOR = #$0F ; API function
API_FN_CURSOR_INVERSE = #$10 ; API function

; Console results (Group 2 Function 2)
API_QUEUE_EMPTY = #$FF ; API result (status code)

; File I/O functions (Group 3)
API_GROUP_FILEIO      = #$03 ; API function group
API_FN_LIST_DIRECTORY = #$01 ; API function
API_FN_LOAD_FILENAME  = #$02 ; API function
API_FN_STORE_FILENAME = #$03 ; API function
API_FN_FILE_OPEN      = #$04 ; API function
API_FN_FILE_CLOSE     = #$05 ; API function
API_FN_FILE_SEEK      = #$06 ; API function
API_FN_FILE_TELL      = #$07 ; API function
API_FN_FILE_READ      = #$08 ; API function
API_FN_FILE_WRITE     = #$09 ; API function
API_FN_FILE_SIZE      = #$0A ; API function
API_FN_FILE_SET_SIZE  = #$0B ; API function
API_FN_FILE_RENAME    = #$0C ; API function
API_FN_FILE_DELETE    = #$0D ; API function
API_FN_DIR_CHDIR      = #$0E ; API function
API_FN_DIR_MKDIR      = #$0F ; API function
API_FN_FILE_STAT      = #$10 ; API function
API_FN_DIR_OPEN       = #$11 ; API function
API_FN_DIR_READ       = #$12 ; API function
API_FN_DIR_CLOSE      = #$13 ; API function
API_FN_FILE_COPY      = #$14 ; API function
API_FN_LIST_FILTERED  = #$20 ; API function

; File I/O parameters (Group 3 Function 2)
API_FILE_TO_SCREEN = #$FFFF ; API parameter

; Mathematics functions (Group 4)
API_GROUP_MATH    = #$04 ; API function group
API_FN_ADD        = #$00 ; API function
API_FN_SUB        = #$01 ; API function
API_FN_MUL        = #$02 ; API function
API_FN_DIV_DEC    = #$03 ; API function
API_FN_DIV_INT    = #$04 ; API function
```

```
API_FN_MOD        = #$05 ; API function
API_FN_COMP       = #$06 ; API function
API_FN_NEG        = #$10 ; API function
API_FN_FLOOR      = #$11 ; API function
API_FN_SQRT       = #$12 ; API function
API_FN_SINE       = #$13 ; API function
API_FN_COS        = #$14 ; API function
API_FN_TAN        = #$15 ; API function
API_FN_ATAN       = #$16 ; API function
API_FN_EXP        = #$17 ; API function
API_FN_LOG        = #$18 ; API function
API_FN_ABS        = #$19 ; API function
API_FN_SIGN       = #$1A ; API function
API_FN_RND_DEC    = #$1B ; API function
API_FN_RND_INT    = #$1C ; API function
API_FN_INT_TO_DEC = #$20 ; API function
API_FN_STR_TO_NUM = #$21 ; API function
API_FN_NUM_TO_STR = #$22 ; API function

; Graphics functions (Group 5)
API_GROUP_GRAPHICS     = #$05 ; API function group
API_FN_SET_GFX         = #$01 ; API function
API_FN_DRAW_LINE       = #$02 ; API function
API_FN_DRAW_RECT       = #$03 ; API function
API_FN_DRAW_ELLIPSE    = #$04 ; API function
API_FN_DRAW_PIXEL      = #$05 ; API function
API_FN_DRAW_TEXT       = #$06 ; API function
API_FN_DRAW_IMG        = #$07 ; API function
API_FN_DRAW_TILEMAP    = #$08 ; API function
API_FN_SET_PALETTE     = #$20 ; API function
API_FN_READ_PIXEL      = #$21 ; API function
API_FN_RESET_PALETTE   = #$22 ; API function
API_FN_SET_TILEMAP     = #$23 ; API function
API_FN_READ_SPRITE_PXL = #$24 ; API function
API_FN_FRAME_COUNT     = #$25 ; API function
API_FN_SET_COLOR       = #$40 ; API function
API_FN_SET_SOLID       = #$41 ; API function
API_FN_SET_DRAW_SIZE   = #$42 ; API function
API_FN_SET_FLIP        = #$43 ; API function

; Graphics parameters (Group 5, Function 1 - Group 6, Function 2)
API_FLIP_HORZ = #$00 ; API parameter (flag)
API_FLIP_VERT = #$01 ; API parameter (flag)

; Graphics results (Group 5, Functions 33,36)
API_PIXEL_TRANSPARENT = #$00 ; API result (flag)

; Sprites functions (Group 6)
API_GROUP_SPRITES       = #$06 ; API function group
API_FN_SPRITE_RESET     = #$01 ; API function
API_FN_SPRITE_SET       = #$02 ; API function
API_FN_SPRITE_HIDE      = #$03 ; API function
API_FN_SPRITE_COLLISION = #$04 ; API function
API_FN_SPRITE_POS       = #$05 ; API function

; Sprites parameters (Group 6, Function 2)
API_SPRITE_TURTLE = #$00 ; API parameter (sprite index)
```

```
API_SPRITE_32BIT  = #$40 ; API parameter (bit-mask)
API_SPRITE_CLEAR  = #$80 ; API parameter (bit-mask)
API_ANCHOR_BL     = #$01 ; API parameter (anchor position)
API_ANCHOR_B      = #$02 ; API parameter (anchor position)
API_ANCHOR_BR     = #$03 ; API parameter (anchor position)
API_ANCHOR_L      = #$04 ; API parameter (anchor position)
API_ANCHOR_C      = #$05 ; API parameter (anchor position)
API_ANCHOR_R      = #$06 ; API parameter (anchor position)
API_ANCHOR_TL     = #$07 ; API parameter (anchor position)
API_ANCHOR_T      = #$08 ; API parameter (anchor position)
API_ANCHOR_TR     = #$09 ; API parameter (anchor position)

; Sprites results (Group 6, Function 4)
API_COLLISION_NONE = #$00 ; API result (flag)

; Controller functions (Group 7)
API_GROUP_CONTROLLER   = #$07 ; API function group
API_FN_READ_CONTROLLER = #$01 ; API function

; Controller results (Group 7, Function 1)
API_CONTROLLER_LEFT  = #$01 ; API result (status bit-mask)
API_CONTROLLER_RIGHT = #$02 ; API result (status bit-mask)
API_CONTROLLER_UP    = #$04 ; API result (status bit-mask)
API_CONTROLLER_DOWN  = #$08 ; API result (status bit-mask)
API_CONTROLLER_BTNA  = #$10 ; API result (status bit-mask)
API_CONTROLLER_BTNB  = #$20 ; API result (status bit-mask)

; Sound functions (Group 8)
API_GROUP_SOUND      = #$08 ; API function group
API_FN_RESET_SOUND   = #$01 ; API function
API_FN_RESET_CHANNEL = #$02 ; API function
API_FN_BEEP          = #$03 ; API function
API_FN_QUEUE_SOUND   = #$04 ; API function
API_FN_PLAY_SOUND    = #$05 ; API function
API_FN_SOUND_STATUS  = #$06 ; API function

; Sound parameters (Group 8, Functions 2,4,5)
API_SOUND_CH_00 = #$00 ; API parameter (channel index)

; Sound parameters (Group 8, Function 4)
API_NOTE_REST      = #$0000 ; API parameter (musical rest)
API_NOTE_C0        = #$0010 ; API parameter (musical note)
API_NOTE_Cs0       = #$0011 ; API parameter (musical note)
API_NOTE_Df0       = #$0011 ; API parameter (musical note)
API_NOTE_D0        = #$0012 ; API parameter (musical note)
API_NOTE_Ds0       = #$0013 ; API parameter (musical note)
API_NOTE_Ef0       = #$0013 ; API parameter (musical note)
API_NOTE_E0        = #$0015 ; API parameter (musical note)
API_NOTE_F0        = #$0016 ; API parameter (musical note)
API_NOTE_Fs0       = #$0017 ; API parameter (musical note)
API_NOTE_Gf0       = #$0017 ; API parameter (musical note)
API_NOTE_G0        = #$0018 ; API parameter (musical note)
API_NOTE_Af0       = #$001A ; API parameter (musical note)
API_NOTE_Gs0       = #$001A ; API parameter (musical note)
API_NOTE_A0        = #$001C ; API parameter (musical note)
API_NOTE_As0       = #$001D ; API parameter (musical note)
API_NOTE_Bf0       = #$001D ; API parameter (musical note)
```

```
API_NOTE_B0        = #$001F ; API parameter (musical note)
API_NOTE_C1        = #$0021 ; API parameter (musical note)
API_NOTE_Cs1       = #$0023 ; API parameter (musical note)
API_NOTE_Df1       = #$0023 ; API parameter (musical note)
API_NOTE_D1        = #$0025 ; API parameter (musical note)
API_NOTE_Ds1       = #$0027 ; API parameter (musical note)
API_NOTE_Ef1       = #$0027 ; API parameter (musical note)
API_NOTE_E1        = #$0029 ; API parameter (musical note)
API_NOTE_F1        = #$002C ; API parameter (musical note)
API_NOTE_Fs1       = #$002E ; API parameter (musical note)
API_NOTE_Gf1       = #$002E ; API parameter (musical note)
API_NOTE_G1        = #$0031 ; API parameter (musical note)
API_NOTE_Af1       = #$0034 ; API parameter (musical note)
API_NOTE_Gs1       = #$0034 ; API parameter (musical note)
API_NOTE_A1        = #$0037 ; API parameter (musical note)
API_NOTE_As1       = #$003A ; API parameter (musical note)
API_NOTE_Bf1       = #$003A ; API parameter (musical note)
API_NOTE_B1        = #$003E ; API parameter (musical note)
API_NOTE_C2        = #$0041 ; API parameter (musical note)
API_NOTE_Cs2       = #$0045 ; API parameter (musical note)
API_NOTE_Df2       = #$0045 ; API parameter (musical note)
API_NOTE_D2        = #$0049 ; API parameter (musical note)
API_NOTE_Ds2       = #$004E ; API parameter (musical note)
API_NOTE_Ef2       = #$004E ; API parameter (musical note)
API_NOTE_E2        = #$0052 ; API parameter (musical note)
API_NOTE_F2        = #$0057 ; API parameter (musical note)
API_NOTE_Fs2       = #$005C ; API parameter (musical note)
API_NOTE_Gf2       = #$005C ; API parameter (musical note)
API_NOTE_G2        = #$0062 ; API parameter (musical note)
API_NOTE_Af2       = #$0068 ; API parameter (musical note)
API_NOTE_Gs2       = #$0068 ; API parameter (musical note)
API_NOTE_A2        = #$006E ; API parameter (musical note)
API_NOTE_As2       = #$0075 ; API parameter (musical note)
API_NOTE_Bf2       = #$0075 ; API parameter (musical note)
API_NOTE_B2        = #$007B ; API parameter (musical note)
API_NOTE_C3        = #$0083 ; API parameter (musical note)
API_NOTE_Cs3       = #$008B ; API parameter (musical note)
API_NOTE_Df3       = #$008B ; API parameter (musical note)
API_NOTE_D3        = #$0093 ; API parameter (musical note)
API_NOTE_Ds3       = #$009C ; API parameter (musical note)
API_NOTE_Ef3       = #$009C ; API parameter (musical note)
API_NOTE_E3        = #$00A5 ; API parameter (musical note)
API_NOTE_F3        = #$00AF ; API parameter (musical note)
API_NOTE_Fs3       = #$00B9 ; API parameter (musical note)
API_NOTE_Gf3       = #$00B9 ; API parameter (musical note)
API_NOTE_G3        = #$00C4 ; API parameter (musical note)
API_NOTE_Af3       = #$00D0 ; API parameter (musical note)
API_NOTE_Gs3       = #$00D0 ; API parameter (musical note)
API_NOTE_A3        = #$00DC ; API parameter (musical note)
API_NOTE_As3       = #$00E9 ; API parameter (musical note)
API_NOTE_Bf3       = #$00E9 ; API parameter (musical note)
API_NOTE_B3        = #$00F7 ; API parameter (musical note)
API_NOTE_C4        = #$0106 ; API parameter (musical note)
API_NOTE_Cs4       = #$0115 ; API parameter (musical note)
API_NOTE_Df4       = #$0115 ; API parameter (musical note)
API_NOTE_D4        = #$0126 ; API parameter (musical note)
API_NOTE_Ds4       = #$0137 ; API parameter (musical note)
```

```
API_NOTE_Ef4       = #$0137 ; API parameter (musical note)
API_NOTE_E4        = #$014A ; API parameter (musical note)
API_NOTE_F4        = #$015D ; API parameter (musical note)
API_NOTE_Fs4       = #$0172 ; API parameter (musical note)
API_NOTE_Gf4       = #$0172 ; API parameter (musical note)
API_NOTE_G4        = #$0188 ; API parameter (musical note)
API_NOTE_Af4       = #$019F ; API parameter (musical note)
API_NOTE_Gs4       = #$019F ; API parameter (musical note)
API_NOTE_A4        = #$01B8 ; API parameter (musical note)
API_NOTE_As4       = #$01D2 ; API parameter (musical note)
API_NOTE_Bf4       = #$01D2 ; API parameter (musical note)
API_NOTE_B4        = #$01EE ; API parameter (musical note)
API_NOTE_C5        = #$020B ; API parameter (musical note)
API_NOTE_Cs5       = #$022A ; API parameter (musical note)
API_NOTE_Df5       = #$022A ; API parameter (musical note)
API_NOTE_D5        = #$024B ; API parameter (musical note)
API_NOTE_Ds5       = #$026E ; API parameter (musical note)
API_NOTE_Ef5       = #$026E ; API parameter (musical note)
API_NOTE_E5        = #$0293 ; API parameter (musical note)
API_NOTE_F5        = #$02BA ; API parameter (musical note)
API_NOTE_Fs5       = #$02E4 ; API parameter (musical note)
API_NOTE_Gf5       = #$02E4 ; API parameter (musical note)
API_NOTE_G5        = #$0310 ; API parameter (musical note)
API_NOTE_Af5       = #$033F ; API parameter (musical note)
API_NOTE_Gs5       = #$033F ; API parameter (musical note)
API_NOTE_A5        = #$0370 ; API parameter (musical note)
API_NOTE_As5       = #$03A4 ; API parameter (musical note)
API_NOTE_Bf5       = #$03A4 ; API parameter (musical note)
API_NOTE_B5        = #$03DC ; API parameter (musical note)
API_NOTE_C6        = #$0417 ; API parameter (musical note)
API_NOTE_Cs6       = #$0455 ; API parameter (musical note)
API_NOTE_Df6       = #$0455 ; API parameter (musical note)
API_NOTE_D6        = #$0497 ; API parameter (musical note)
API_NOTE_Ds6       = #$04DD ; API parameter (musical note)
API_NOTE_Ef6       = #$04DD ; API parameter (musical note)
API_NOTE_E6        = #$0527 ; API parameter (musical note)
API_NOTE_F6        = #$0575 ; API parameter (musical note)
API_NOTE_Fs6       = #$05C8 ; API parameter (musical note)
API_NOTE_Gf6       = #$05C8 ; API parameter (musical note)
API_NOTE_G6        = #$0620 ; API parameter (musical note)
API_NOTE_Af6       = #$067D ; API parameter (musical note)
API_NOTE_Gs6       = #$067D ; API parameter (musical note)
API_NOTE_A6        = #$06E0 ; API parameter (musical note)
API_NOTE_As6       = #$0749 ; API parameter (musical note)
API_NOTE_Bf6       = #$0749 ; API parameter (musical note)
API_NOTE_B6        = #$07B8 ; API parameter (musical note)
API_NOTE_C7        = #$082D ; API parameter (musical note)
API_NOTE_Cs7       = #$08A9 ; API parameter (musical note)
API_NOTE_Df7       = #$08A9 ; API parameter (musical note)
API_NOTE_D7        = #$092D ; API parameter (musical note)
API_NOTE_Ds7       = #$09B9 ; API parameter (musical note)
API_NOTE_Ef7       = #$09B9 ; API parameter (musical note)
API_NOTE_E7        = #$0A4D ; API parameter (musical note)
API_NOTE_F7        = #$0AEA ; API parameter (musical note)
API_NOTE_Fs7       = #$0B90 ; API parameter (musical note)
API_NOTE_Gf7       = #$0B90 ; API parameter (musical note)
API_NOTE_G7        = #$0C40 ; API parameter (musical note)
```

```
API_NOTE_Af7       = #$0CFA ; API parameter (musical note)
API_NOTE_Gs7       = #$0CFA ; API parameter (musical note)
API_NOTE_A7        = #$0DC0 ; API parameter (musical note)
API_NOTE_As7       = #$0E91 ; API parameter (musical note)
API_NOTE_Bf7       = #$0E91 ; API parameter (musical note)
API_NOTE_B7        = #$0F6F ; API parameter (musical note)
API_NOTE_C8        = #$105A ; API parameter (musical note)
API_NOTE_Cs8       = #$1153 ; API parameter (musical note)
API_NOTE_Df8       = #$1153 ; API parameter (musical note)
API_NOTE_D8        = #$125B ; API parameter (musical note)
API_NOTE_Ds8       = #$1372 ; API parameter (musical note)
API_NOTE_Ef8       = #$1372 ; API parameter (musical note)
API_NOTE_E8        = #$149A ; API parameter (musical note)
API_NOTE_F8        = #$15D4 ; API parameter (musical note)
API_NOTE_Fs8       = #$1720 ; API parameter (musical note)
API_NOTE_Gf8       = #$1720 ; API parameter (musical note)
API_NOTE_G8        = #$1880 ; API parameter (musical note)
API_NOTE_Af8       = #$19F5 ; API parameter (musical note)
API_NOTE_Gs8       = #$19F5 ; API parameter (musical note)
API_NOTE_A8        = #$1B80 ; API parameter (musical note)
API_NOTE_As8       = #$1D23 ; API parameter (musical note)
API_NOTE_Bf8       = #$1D23 ; API parameter (musical note)
API_NOTE_B8        = #$1EDE ; API parameter (musical note)
API_NOTE_C9        = #$20B4 ; API parameter (musical note)
API_NOTE_Cs9       = #$22A6 ; API parameter (musical note)
API_NOTE_Df9       = #$22A6 ; API parameter (musical note)
API_NOTE_D9        = #$24B5 ; API parameter (musical note)
API_NOTE_Ds9       = #$26E4 ; API parameter (musical note)
API_NOTE_Ef9       = #$26E4 ; API parameter (musical note)
API_NOTE_E9        = #$2934 ; API parameter (musical note)
API_NOTE_F9        = #$2BA7 ; API parameter (musical note)
API_NOTE_Fs9       = #$2E40 ; API parameter (musical note)
API_NOTE_Gf9       = #$2E40 ; API parameter (musical note)
API_NOTE_G9        = #$3100 ; API parameter (musical note)
API_NOTE_Af9       = #$33EA ; API parameter (musical note)
API_NOTE_Gs9       = #$33EA ; API parameter (musical note)
API_NOTE_A9        = #$3700 ; API parameter (musical note)
API_NOTE_As9       = #$3A45 ; API parameter (musical note)
API_NOTE_Bf9       = #$3A45 ; API parameter (musical note)
API_NOTE_B9        = #$3DBC ; API parameter (musical note)
API_NOTE_C10       = #$4168 ; API parameter (musical note)
API_NOTE_Cs10      = #$454C ; API parameter (musical note)
API_NOTE_Df10      = #$454C ; API parameter (musical note)
API_NOTE_D10       = #$496B ; API parameter (musical note)
API_NOTE_Ds10      = #$4DC8 ; API parameter (musical note)
API_NOTE_Ef10      = #$4DC8 ; API parameter (musical note)
API_TEMPO_60       = #$0064 ; API parameter (musical note duration, 60BPM)
API_TEMPO_80       = #$004B ; API parameter (musical note duration, 80BPM)
API_TEMPO_90       = #$0042 ; API parameter (musical note duration, 90BPM)
API_TEMPO_120      = #$0032 ; API parameter (musical note duration, 120BPM)
API_SLIDE_NONE     = #$0000 ; API parameter (slide value)
API_SLIDE_SLOW     = #$0004 ; API parameter (slide range)
API_SLIDE_MED      = #$0008 ; API parameter (slide range)
API_SLIDE_FAST     = #$0010 ; API parameter (slide range)
API_SOUND_SRC_BEEP = #$00   ; API parameter (sound generator)

; Sound parameters (Group 8, Function 5)
```

```
API_SFX_POSITIVE     = #$00 ; API parameter (sound effect)
API_SFX_NEGATIVE     = #$01 ; API parameter (sound effect)
API_SFX_ERROR        = #$02 ; API parameter (sound effect)
API_SFX_CONFIRM      = #$03 ; API parameter (sound effect)
API_SFX_REJECT       = #$04 ; API parameter (sound effect)
API_SFX_SWEEP        = #$05 ; API parameter (sound effect)
API_SFX_COIN         = #$06 ; API parameter (sound effect)
API_SFX_LASER_LONG   = #$07 ; API parameter (sound effect)
API_SFX_POWERUP      = #$08 ; API parameter (sound effect)
API_SFX_VICTORY      = #$09 ; API parameter (sound effect)
API_SFX_DEFEAT       = #$0A ; API parameter (sound effect)
API_SFX_FANFARE      = #$0B ; API parameter (sound effect)
API_SFX_ALARM1       = #$0C ; API parameter (sound effect)
API_SFX_ALARM2       = #$0D ; API parameter (sound effect)
API_SFX_ALARM3       = #$0E ; API parameter (sound effect)
API_SFX_RING1        = #$0F ; API parameter (sound effect)
API_SFX_RING2        = #$10 ; API parameter (sound effect)
API_SFX_RING3        = #$11 ; API parameter (sound effect)
API_SFX_DANGER       = #$12 ; API parameter (sound effect)
API_SFX_EXPL_LONG    = #$13 ; API parameter (sound effect)
API_SFX_EXPL_MEDIUM  = #$14 ; API parameter (sound effect)
API_SFX_EXPL_SHORT   = #$15 ; API parameter (sound effect)
API_SFX_LASER_MEDIUM = #$16 ; API parameter (sound effect)
API_SFX_LASER_SHORT  = #$17 ; API parameter (sound effect)

; Turtle Graphics functions (Group 9)
API_GROUP_TURTLE   = #$09 ; API function group
API_FN_TURTLE_INIT = #$01 ; API function
API_FN_TURTLE_TURN = #$02 ; API function
API_FN_TURTLE_MOVE = #$03 ; API function
API_FN_TURTLE_HIDE = #$04 ; API function
API_FN_TURTLE_HOME = #$05 ; API function

; Turtle Graphics parameters (Group 9, Function 2)
API_TURTLE_LEFT  = #$010E ; API parameter (turn -90 degrees)
API_TURTLE_RIGHT = #$005A ; API parameter (turn +90 degrees)
API_TURTLE_FLIP  = #$00B4 ; API parameter (turn 180 degrees)

; Turtle Graphics parameters (Group 9, Function 3)
API_PEN_UP   = #$00 ; API parameter (turtle tracks on)
API_PEN_DOWN = #$01 ; API parameter (turtle tracks off)

; UExt functions (Group 10)
API_GROUP_UEXT      = #$09 ; API function group
API_FN_UEXT_INIT    = #$01 ; API function
API_FN_GPIO_WRITE   = #$02 ; API function
API_FN_GPIO_READ    = #$03 ; API function
API_FN_SET_PORT_DIR = #$04 ; API function
API_FN_I2C_WRITE    = #$05 ; API function
API_FN_I2C_READ     = #$06 ; API function
API_FN_ANALOG_READ  = #$07 ; API function


;--------;
; colors ;
;--------;
```

```
COLOR_BLACK       = #$80
COLOR_RED         = #$81
COLOR_GREEN       = #$82
COLOR_YELLOW      = #$83
COLOR_BLUE        = #$84
COLOR_MAGENTA     = #$85
COLOR_CYAN        = #$86
COLOR_WHITE       = #$87
COLOR_ALT_BLACK   = #$88
COLOR_DARK_GREY   = #$89
COLOR_DARK_GREEN  = #$8A
COLOR_ORANGE      = #$8B
COLOR_DARK_ORANGE = #$8C
COLOR_BROWN       = #$8D
COLOR_PINK        = #$8E
COLOR_LIGHT_GREY  = #$8F
```

Pascal for the Neo6502

Appendix A

This appendix offers the detailed specifications for each of the Neo6502 computers and the differences between them.

Feature	Neo6502 Computer	Neo6502pc Computer
Physical W65C02 processor running at 6.25Mhz	Yes	Yes
RP2040 SoC (System on a Chip) w/ 2MB Flash	Yes	Yes
10-pin UEXT connector	1x	4x
6502 bus connector	Yes	Yes
Audio mini speaker	Yes	Yes
Audio 3.5mm connector	Yes	Yes
USB-C power supply connector	Yes	Yes
DVI/HDMI connector	Yes	No‡
USB-A Port§	1x**	3x
4-position configuration slide switch	Yes	Yes
Boot Button	Yes	Yes
One 14-pin external 12 GPIO connector	No	Yes
Programming slide switch	No	Yes
USB-C Programming Port	No	Yes
Build-in LCD display (320x240) w/ touch panel acting like mouse	No	Yes
Build in LiPo battery w/ charger circuit and battery monitoring	No	Yes
Power on off switch	No	Yes

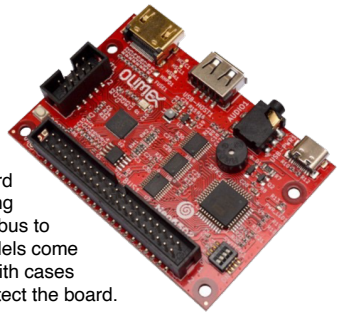
‡ The HDMI port is utilized by the built-in LCD display.

§ The USB-A port can be used for various USB accessories (keyboards, flash drives, gamepad, etc..) and for the Neo6502 – it serves as the RP2040 programming port (*requires a USB-A to USB-A cable*).

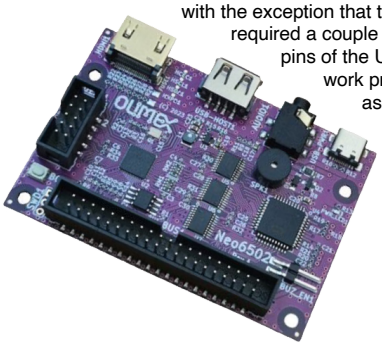
** Requires the USB-Neohub (https://www.olimex.com/Products/USB-Modules/USB-NeoHub/open-source-hardware) the expand the number of USB-A ports available. Additional ports are required to utilize NeoBasic, as you need at minimum a flash drive and keyboard. *Unlike the USB-Neohub, not all USB hubs are compatible or supported.*

Neo6502

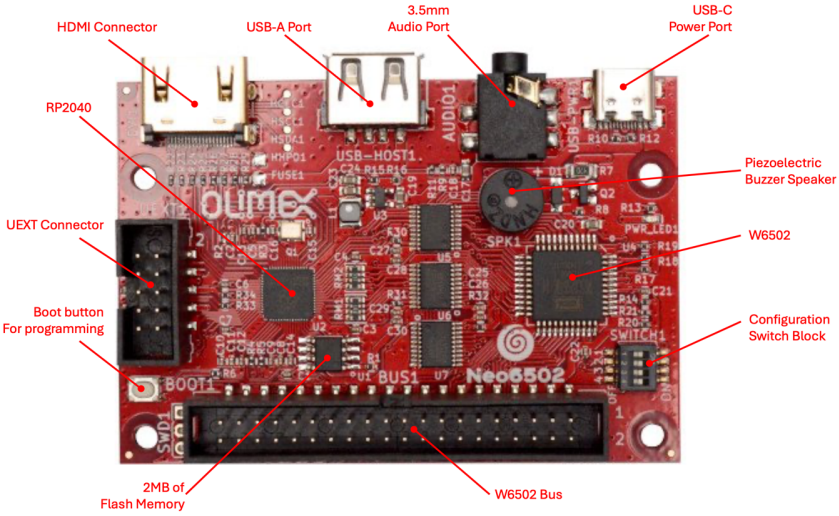
The Olimex Neo6502 was the first model released. The main components are a W65C02 and a Raspberry Pi RP2040. The W65C02 runs the machine code (at about 6.3Mhz), with the RP2040 providing RAM, Video and other aspects.



Early adopters had a revision A board (purple), and later the revision B board (red) was released. Both are almost identical, with the exception that the revision A board required a couple of wires connecting pins of the UEXT to the 6502 bus to work properly. Both models come as the board only, with cases available to protect the board.



Hardware Pictures



Neo6502pc

The Olimex Neo6502pc is an open-source hardware and software standalone modern retro computer with a real W65C02 processor, RP2040 co-processor, USB host (with 3x USB-A ports), Lithium polymer (Lipo) battery with charging circuit and battery monitoring.

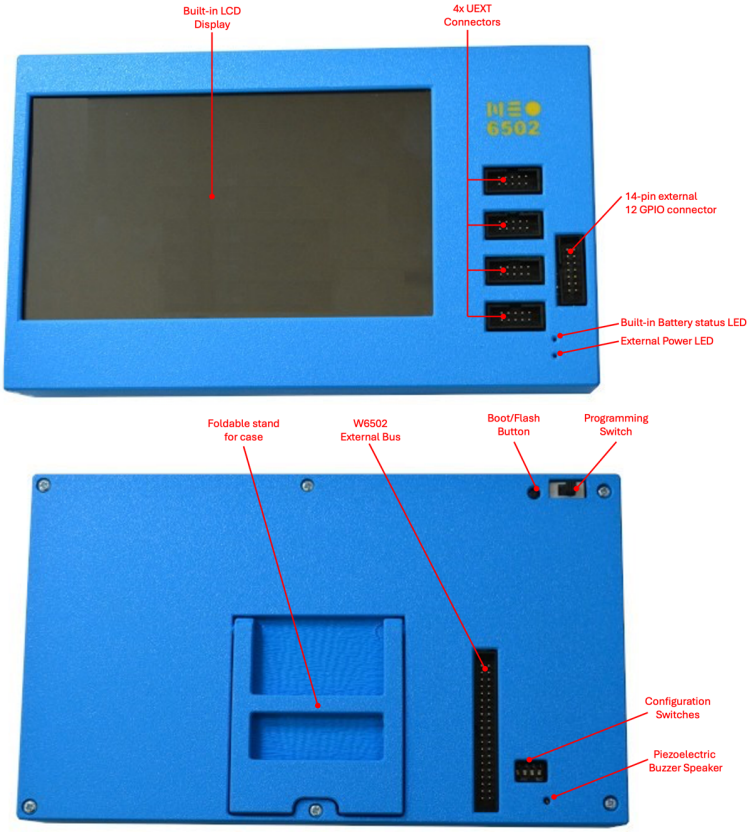
The design goal with Neo6502pc was to create a simple retro-computer with a real 6502 processor that has modern features (provided by the Raspberry RP2040 co-processor) such as HDMI video output, USB ports that support USB keyboards, flash drives for storage, and USB game pad. The RP2040 co-processor emulates the RAM for the W6502 processor.

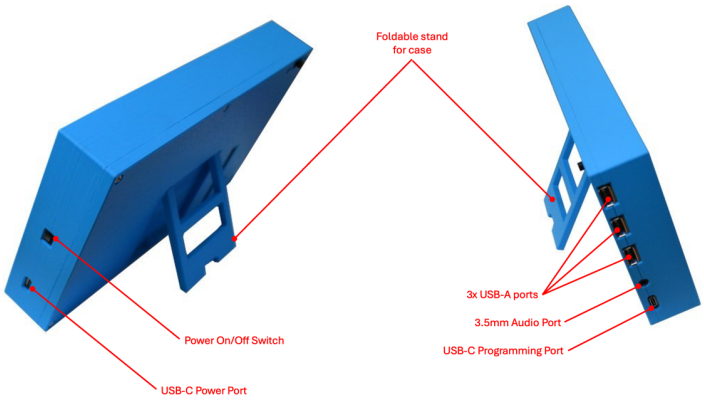
More Information: https://www.olimex.com/Products/Retro-Computers/Neo6502pc/open-source-hardware

Features

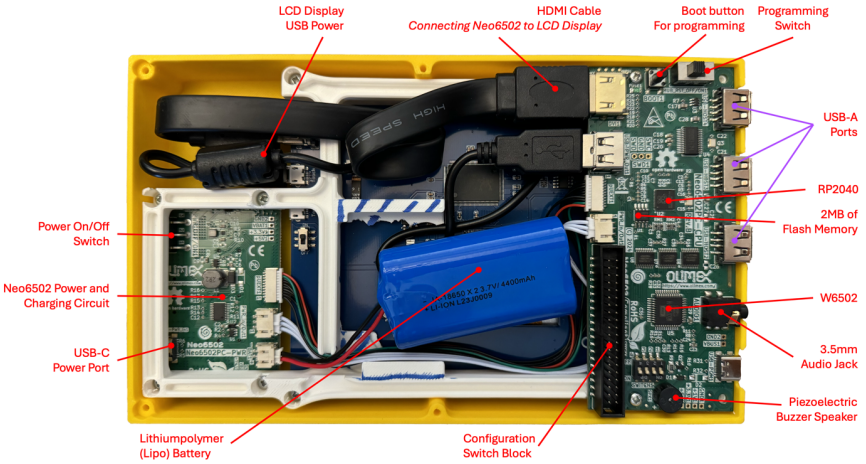
- A real W65C02 processor clocked at 6.25Mhz
- Graphics co-processor RP2040 providing 320 x 240 resolution with 256-color display on HDMI/DVI.
- 32k Graphics RAM for tiles and sprites
- 128 sprites up to 32x32 pixels.
- Multiple tile maps (16x16 tiles, can be double sized)
- High speed drawing features
- Turtle Graphics
- Blitter for high-speed graphics
- Four UEXT interface ports to access a wide range of hardware add ons.
- 1 channel "beeper" sound with SFX library (to be replaced by AY-3-8910 Emulation)
- USB flash drive support for storage w/ optional SD Card support.
- Supports standard USB keyboard.
- Fast structured BASIC with hardware support and inline assembler.
- BASIC can be edited on screen or using a text editor.
- High Speed Integer/Floating point arithmetic
- Huge open-source community that has written documentation, provided samples, and code including games.
- Cross development support
- Accurate cross platform emulator for Windows/Mac/Linux, only requires SDL2
- Serial link to PC for Cross-Development
- Program in PASCAL using Mad Pascal compiler
- Program in 'C' using CC65 and LLVM
- USB Mouse and Gamepad support
- BASIC support for Serial, I2C and SPI hardware via UEXT Connector - 64KB linear RAM space for code
- LCD display
- Internal battery backup power supply which allows it to operate up to 3 hours without external power supply
- Three external and one internal USB hosts (internal is connected to LCD touch panel)
- Audio output
- 12 GPIO extension connector
- USB-C for power and internal battery charging.
- Second USB-C for RP2040 firmware programming
- Dimensions 220 x 130 x 35 mm

Neo6502pc – Hardware Pictures




Foldable stand
for case
Power On/Off Switch
USB-C Power Port
3x USB-A ports
3.5mm Audio Port
USB-C Programming Port

Neo6502pc – Hardware Pictures *(continued)*



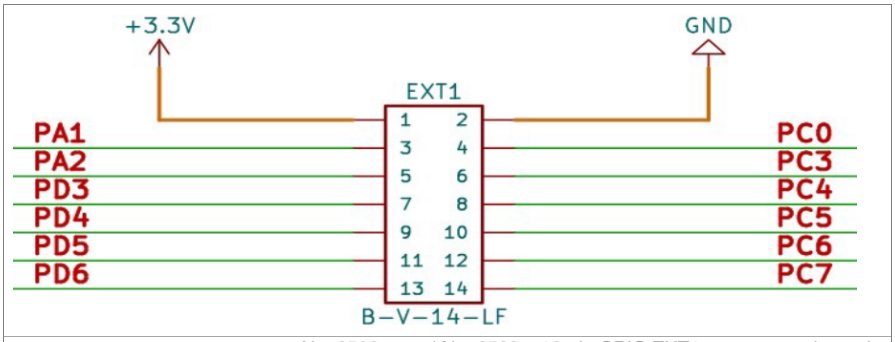
Neo6502pc Specific Hardware Specifications

Neo6502pc – Schematic

The latest schematic for the Neo6502pc is available on GitHub using this link:
https://github.com/OLIMEX/Neo6502pc

Neo6502pc – 12 GPIO EXT1 Connector

Neo6502pc has a CH32V003 expander IC which is connected to RP2040 via I2C and can monitor battery charge, the presence of the external power supply and access to the 12 GPIOs via the EXT1 connector:



Neo6502pc and Neo6502 – 12-pin GPIO EXT1 connector schematic

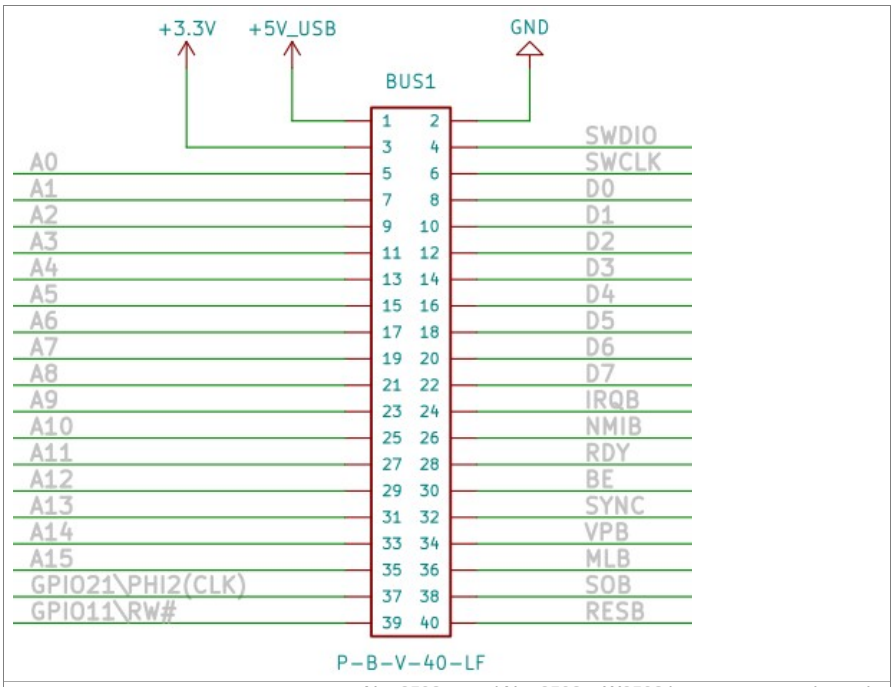
Shared Hardware

Neo6502pc and Neo6502 – W6502 Bus Connector

All 6502 signals are available on BUS1 connector for attaching external hardware on it. Signals available:

- +5V
- 3.3V
- GND
- D0-D7
- A0-A15
- PHI2
- R/W
- RESB
- SOB
- MLB
- VPB
- SYNC
- NMIB
- IRQB

Two signals of RP2040 SWDIO and SWCLK are also present for RP2040 debugging, these should not be connected on the external 6502 peripheral boards.



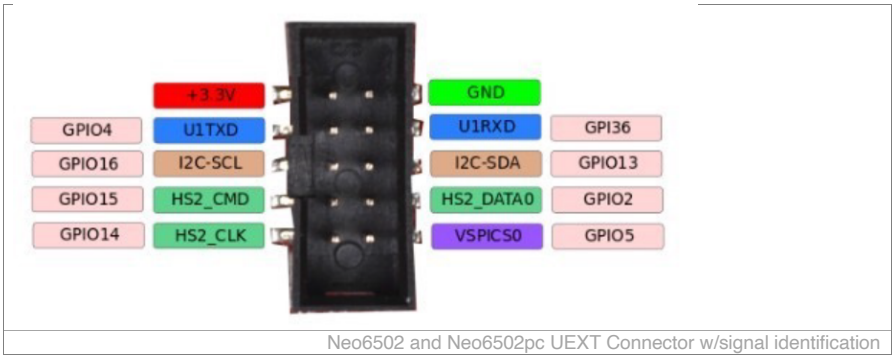
Neo6502pc and Neo6502 – W6502 bus connector schematic

Neo6502pc and Neo6502 – UEXT Connectors

UEXT (Universal EXTension) connectors have the following signals available. All signals are with 3.3V levels.

- +3.3V
- GND
- I2C
- SPI
- UART

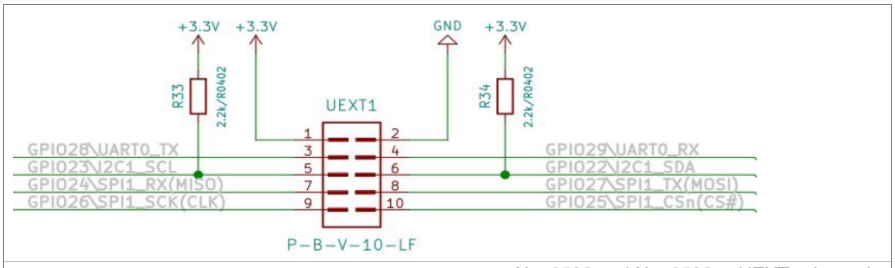
UEXT connector can be found in many different shapes, however the connector used on the Neo6502pv uses a UEXT connector that is 0.1" 2.54mm step boxed plastic connector.



Neo6502 and Neo6502pc UEXT Connector w/signal identification

Olimex has developed number of MODULES with this connector: temperature, humidity, pressure, magnetic field, light sensors, LCDs, LED matrix, Relays, Bluetooth, Zigbee, WiFi, GSM, GPS, RFID, RTC, EKG, sensors and etc.

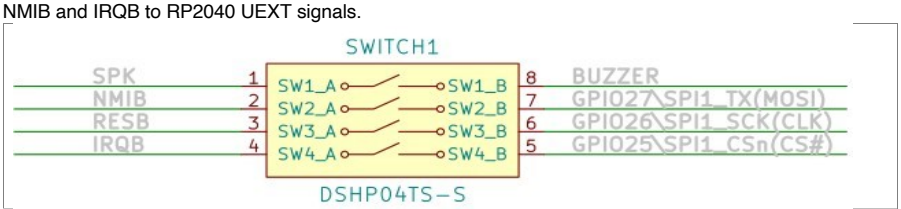
Neo6502pc UEXT connector is wired to RP2040 GPIOs as follows:



Neo6502 and Neo6502pc UEXT schematic

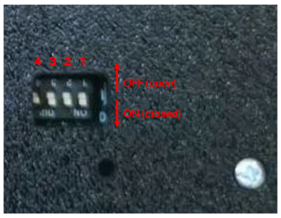
Neo6502pc and Neo6502 – Configuration Switch Block

The Slide configuration switch can enable/disable the Buzzer, also can connect or disconnect RESB, NMIB and IRQB to RP2040 UEXT signals.



Neo6502 and Neo6502pc configuration switch block schematic

By default, the Neo6502pc is shipped with all switches in the closed state – enabling the Piezoelectric Buzzer Speaker, and all signals wired to the RP2040. With SW2, SW3 and SW4 enabled, the SPI on UEXT cannot be used.



Appendix B – CREDITS and LICENSE

The Neo6502 board, schematics, and firmware are all part of an Open Source project created by Mr. Tsvetan Usunov of Bulgaria.

This document initially created by

Greg Jewett
Email: greg(at)ejewett(dot)com
Website: https://sites.google.com/ejewett.com/gregjewett/home

For a list of contributing authors and updates, please visit the Github repository:
https://github.com/jewettg/Neo6502-Documentation/blob/main/CONTRIBUTING.md

Appendix C – Document Revision History

Please visit the GitHub repository file CHANGELOG.md
https://github.com/jewettg/Neo6502-Documentation/blob/main/CHANGELOG.md

Appendix D – About Olimex

OLIMEX Ltd.
2 Pravda St., P.O. Box 237,
Plovdiv 4000 BULGARIA

Contact: Mr. Tsvetan Usunov
Email: info@olimex.com
Voice: +359-32-626259, +359-32-267407, +359-32-621270

Olimex Ltd is a leading provider for development tools and programmers for embedded market.

The company has 28 years of experience in designing, prototyping and manufacturing printed circuit boards, sub-assemblies, and complete electronic products.

We were established in 1991 in Plovdiv - the second largest city in Bulgaria.

We have extensive knowledge in analog, digital, and microcontroller design, and we offer our own-designed development boards, programmers and emulators for rapid prototyping ARM, AVR, MSP430, MAXQ and PIC microcontrollers.

Olimex is recognized as an approved third-party hardware developer by Texas Instruments Inc., Maxim Integrated, Atmel Inc., NXP Inc., ST Microelectronics Inc., IAR Systems AB, Cirrus Logic Inc., OKI Semiconductor Inc, Energy Micro Inc., and Microchip Inc.

We have over 30,000 active customer accounts who regularly use our services for electronic boards development and prototyping. Our design capabilities are backed by our own PCB prototype production and assembly facility, so all designs made by us are created with design-for-manufacturing in mind - which guarantees that they are optimized for reliability and provide cost-effective solutions for our customers.

The company's 5,000 sq m. production buildings are situated on our 10,000 sq m. property.

Appendix E – Online Resources

As an open-source project, there are a ton of resources already available for the Neo6502! Below is a list (not exhaustive, and growing) of various websites, repositories, and more.

If you find a link that no longer works, please let us know.

Purchasing Neo6502	• Tindie (Australia) • The Pi Hut (UK) • Digikey (US)
GitHub	**Neo6502:** Olimex Neo6502 Github Repository **Documentation:** Neo6502-Documentation
OLIMEX	**Neo6502 Project:** Official Product Website **UEXT Modules:** UEXT Modules available from Olimex
Facebook	Facebook Page
Discord	Discord Server