CNC4FUN
GRBL-1.1F - IW2EAX-1.1.41 Configuration

From: https://github.com/gnea/grbl/wiki/Grbl-v1.1-Configuration

Original manual by Charles Van Noland, to which we added our own extended commands (in yellow), which are available on the CNC4FUN (by IW2EAX) controller board.

Getting Started

First, connect to Grbl using the serial terminal of your choice.

Set the baud rate to **115200** as 8-N-1 (8-bits, no parity, and 1-stop bit).

Once connected you should get the Grbl-prompt, which looks like this:

```
Grbl 1.1f ['$' for help]
```

Type $ and press enter to have Grbl print a help message. You should not see any local echo of the $ and enter. Grbl should respond with:

```
[HLP:$$ $# $G $I $N $x=val $Nx=line $J=line $SLP $C $X $H ~ ! ? ctrl-x]
```

The '$'-commands are Grbl system commands used to tweak the settings, view or change Grbl's states and running modes, and start a homing cycle. The last four **non**-'$' commands are realtime control commands that can be sent at anytime, no matter what Grbl is doing. These either immediately change Grbl's running behavior or immediately print a report of the important realtime data like current position (aka DRO).

Grbl Settings

$$ - View Grbl settings

To view the settings, type $$ and press enter after connecting to Grbl. Grbl should respond with a list of the current system settings, as shown in the example below. All of these settings are persistent and kept in EEPROM, so if you power down, these will be loaded back up the next time you power up your Arduino.

The x of $x=val indicates a particular setting, while val is the setting value. In prior versions of Grbl, each setting had a description next to it in () parentheses, but Grbl v1.1+ no longer includes them unfortunately. This was done to free up precious flash memory to add the new features available in v1.1. However, most good GUIs will help out by attaching descriptions for you, so you know what you are looking at.

**Parameters added by cnc4fun*

Settings and sample values	Description
$0=10	Step pulse, microseconds
$1=250	Step idle delay, milliseconds
$2=0	Step port invert, mask
$3=0	Direction port invert, mask
$4=0	Step enable invert, boolean
$5=0	Limit pins invert, boolean
$6=0	Probe pin invert, boolean
$10=1	Status report, mask
$11=0.010	Junction deviation, mm
$12=0.002	Arc tolerance, mm
$13=0	Report inches, boolean
$20=0	Soft limits, boolean (modified behavioural)
$21=0	Hard limits, boolean
$22=1	Homing cycle, boolean
$23=0	Homing dir invert, mask
$24=25.000	Homing feed, mm/min
$25=500.000	Homing seek, mm/min
$26=250	Homing debounce, milliseconds
$27=1.000	Homing pull-off, mm
$30=1000.	Max spindle speed, RPM
$31=0.	Min spindle speed, RPM
$32=0	Laser mode, boolean
$33 = 0	MIST-ON only when Z axis is down
$50 = 1	Expansion board installed
$51 = 1	Pendant enable
$52 = 1	Spindle off on 0 speed
$53 = -1.000	Parking target

Settings and sample values	Description
$54 = 500.000	Parking rate
$55 = 100.000	Parking pullout rate
$56 = 5.000	Parking pullout increment
$57 = 18000	Spindle PWM frequency
$58 = 1500	Servo wake up time
$59 = 7	Homing cycle, mask (XYZ)
$60 = 0	Spindle knob ADC min value
$61 = 372	Spindle knob ADC mid value
$62 = 396	Spindle knob ADC high value
$63 = 0.7	Spindle knob min multiplier
$64 = 1.3	Spindle knob max multiplier
$65 = 32	Spindle knob nr. of steps
$66 = 0	Spindle spin-up time (mSec)
$70 = 0	Feed knob ADC min value
$71 = 372	Feed knob ADC mid value
$72 = 744	Feed knob ADC high value
$73 = 0.10	Feed knob min multiplier
$74 = 3.00	Feed knob max multiplier
$75 = 32	Feed knob nr. of steps
$76 = 1	Feed knob hold at 0 speed
$77 = 1.0	Feed minimum speed
$80 = 1.5	Pendant XY fast-mode timer
$81 = 1.5	Pendant Z fast-mode timer
$82 = 3500.00	Pendant XY jog-feed speed
$83 = 1000.00	Pendant Z jog-feed speed
$84 = 0.010	Pendant jog-step when pushbuttons = 00
$85 = 0.100	Pendant jog-step when pushbuttons = 01
$86 = 1.000	Pendant jog-step when pushbuttons = 10
$87 = 10.000	Pendant jog-step when pushbuttons = 11
$88 = 2	Pendant swap mask for XYZ pushbuttons
$89 = 4.00	Pendant soft reset = REBOOT after xx seconds
$90 = 0	Force WiFi off
$91 = 115200	Serial port Baud-rate
$100=250.000	X steps/mm
$101=250.000	Y steps/mm
$102=250.000	Z steps/mm

Settings and sample values	Description
$110=500.000	X Max rate, mm/min
$111=500.000	Y Max rate, mm/min
$112=500.000	Z Max rate, mm/min
$120=10.000	X Acceleration, mm/sec^2
$121=10.000	Y Acceleration, mm/sec^2
$122=10.000	Z Acceleration, mm/sec^2
$130=200.000	X Max travel, mm
$131=200.000	Y Max travel, mm
$132=200.000	Z Max travel, mm

@ - New way to view and change Grbl settings – added by cnc4fun

The @ command is similar to the $$ command but, unlike $$ which is just a list of numbers, it responds with a more structured list of the relevant parameters and descriptions. Please note that the settings are organized by function and not by number.

Moreover, unlike $$, which only provides the complete list of parameters, the @ command can provide just a single parameter: just add the wanted number after the @ and this will happen.

@0 will respond with the single parameter

```
$0   =       3  STEPPER  - Pulse, microseconds
```

@ will respond with the full list

```
-----------------------------------------------------
$0   =       3  STEPPER  - Pulse, microseconds
$1   =     250  STEPPER  - Idle delay, milliseconds
$2   =       0  PIN      - Port invert, mask
$3   =       0  PIN      - Direction port invert, mask
$4   =       0  PIN      - Step enable invert, boolean
$5   =       0  PIN      - Limit invert, boolean
$6   =       0  PIN      - Probe pin invert, boolean
$58 =     1500  SERVO    - Wakeup time (ms)
-----------------------------------------------------
$100 = 2560.00 X-AXE     - Steps/mm
$101 = 2560.00 Y-AXE     - Steps/mm
$102 = 2560.00 Z-AXE     - Steps/mm
-----------------------------------------------------
$110 =  380.00 X-AXE     - Max rate, mm/min
$111 =  380.00 Y-AXE     - Max rate, mm/min
```

```
$112 =  380.00 Z-AXE    - Max rate, mm/min
-----------------------------------------------------
$120 =   80.00 X-AXE    - Acceleration, mm/sec^2
$121 =   80.00 Y-AXE    - Acceleration, mm/sec^2
$122 =   80.00 Z-AXE    - Acceleration, mm/sec^2
-----------------------------------------------------
$130 =  220.00 X-AXE    - Max travel, mm
$131 =  220.00 Y-AXE    - Max travel, mm
$132 =  220.00 Z-AXE    - Max travel, mm
-----------------------------------------------------
$20 =       0  Homing   - Soft limits, boolean
$21 =       1  Homing   - Hard limits, boolean
$22 =       1  Homing   - Enable, boolean
$59 =       7  Homing   - Enable, mask (XYZ)
$23 =       1  Homing   - DIR invert, mask
$24 =   60.00  Homing   - Feed, mm/min
$25 =  250.00  Homing   - Seek, mm/min
$26 =      10  Homing   - Debounce, milliseconds
$27 =    1.00  Homing   - Pull-off, mm
-----------------------------------------------------
$53 =   -1.00  Parking  - Target
$54 =  500.00  Parking  - Rate
$55 =  100.00  Parking  - Pullout rate
$56 =    5.00  Parking  - Pullout increment
-----------------------------------------------------
$10 =       3  GRBL     - REPORT Status, mask
$13 =       0  GRBL     - REPORT Inches, boolean
$11 =   0.010  GRBL     - Junction deviation, mm
$12 =   0.002  GRBL     - Arc tolerance, mm
$32 =       0  GRBL     - Laser mode, boolean
$33 =       0  GRBL     - MIST-ON only when Z axis is down
$91 =  115200  GRBL     - Serial port Baud-rate
-----------------------------------------------------
$50 =       1  PENDANT  - EXP.Board is installed,(reboot)
$51 =       1  PENDANT  - Pendant enable, boolean
$80 =    1.50  PENDANT  - XY Fast-Mode timer (Sec.)
$81 =    1.50  PENDANT  -  Z Fast-Mode timer (Sec.)
$82 = 3500.00  PENDANT  - XY JOG-FEED Speed
$83 = 1000.00  PENDANT  -  Z JOG-FEED Speed
$84 =   0.010  PENDANT  - JOG-STEP when pushbuttons = 00
$85 =   0.100  PENDANT  - JOG-STEP when pushbuttons = 01
$86 =   1.000  PENDANT  - JOG-STEP when pushbuttons = 10
$87 =  10.000  PENDANT  - JOG-STEP when pushbuttons = 11
$88 =       2  PENDANT  - Swap mask for XYZ pushbuttons
$89 =     4.0  PENDANT  - Soft reset = REBOOT after xx sec.
-----------------------------------------------------
$57 =   18000  SPINDLE  - PWM frequency in Hz,(reboot)
$52 =       1  SPINDLE  - OFF on 0 speed, boolean
$30 =   10000  SPINDLE  - Max speed, RPM
$31 =       0  SPINDLE  - Min speed, RPM
```

```
$60 =       0  SPINDLE  - Knob ADC min value
$61 =     372  SPINDLE  - Knob ADC mid value
$62 =     396  SPINDLE  - Knob ADC high value
$63 =     0.7  SPINDLE  - Knob min multiplier
$64 =     1.3  SPINDLE  - Knob max multiplier
$65 =      32  SPINDLE  - Knob Nr.of steps
$66 =    1500  SPINDLE  - Spindle spin-up time (mSec.)
@69 = ADC Readings now: - ADC=339 Scaled=45% Knob=1.0
-----------------------------------------------------
$70 =       0  FEED     - Knob ADC min value
$71 =     372  FEED     - Knob ADC mid value
$72 =     744  FEED     - Knob ADC high value
$73 =     0.1  FEED     - Knob min multiplier
$74 =     3.0  FEED     - Knob max multiplier
$75 =      32  FEED     - Knob Nr.of steps
$76 =       1  FEED     - Knob Hold at 0 Speed
$77 =     1.0  FEED     - Minimum Speed
@79 = ADC Readings now: - ADC=389 Scaled=52% Knob=1.1
-----------------------------------------------------
$90 =       0  WIFI     - Force WIFI OFF
```

Please note:

- The two lines @69 and @79 are read-only values that can be used to calibrate the virtual knobs.

- To set a specific value for all other parameters, you can either just keep using the old $x=val command or a new @x=val command.

NOTE2: @R is a special function that will perform a system reboot.

$x=val - Save Grbl setting

The $x=val command saves or alters a Grbl setting, which can be done manually by sending this command when connected to Grbl through a serial terminal program, but most Grbl GUIs will do this for you as a user-friendly feature.

To manually change e.g. the microseconds step pulse option to 10us you would type this, followed by an enter:

```
$0=10
```

If everything went well, Grbl will respond with an 'ok' and this setting is stored in EEPROM and will be retained forever or until you change them. You can check if Grbl has received and stored your setting correctly by typing $$ to view the system settings again.

Grbl's $x=val settings and what they mean

NOTE: From Grbl v0.9 to Grbl v1.1, only $10 status reports changed and new $30/ $31 spindle rpm max/min and $32 laser mode settings were added. Everything else is the same.

$0 – Step pulse, microseconds

Stepper drivers are rated for a certain minimum step pulse length. Check the data sheet or just try some numbers. You want the shortest pulses the stepper drivers can reliably recognize. If the pulses are too long, you might run into trouble when running the system at very high feed and pulse rates, because the step pulses can begin to overlap each other. We recommend something around 10 microseconds, which is the default value.

$1 - Step idle delay, milliseconds

Every time your steppers complete a motion and come to a stop, Grbl will delay disabling the steppers by this value. **OR**, you can always keep your axes enabled (powered so as to hold position) by setting this value to the maximum 255 milliseconds. Again, just to repeat, you can keep all axes always enabled by setting $1=255.

The stepper idle lock time is the time length Grbl will keep the steppers locked before disabling. Depending on the system, you can set this to zero and disable it. On others, you may need 25-50 milliseconds to make sure your axes come to a complete stop before disabling. This is to help account for machine motors that do not like to be left on for long periods of time without doing something. Also, keep in mind that some stepper drivers don't remember which micro step they stopped on, so when you re-enable, you may witness some 'lost' steps due to this. In this case, just keep your steppers enabled via $1=255.

$2 – Step port invert, mask

This setting inverts the step pulse signal. By default, a step signal starts at normal-low and goes high upon a step pulse event. After a step pulse time set by $0, the pin resets to low, until the next step pulse event. When inverted, the step pulse behavior switches from normal-high, to low during the pulse, and back to high. Most users will not need to use this setting, but this can be useful for certain CNC-stepper drivers that have peculiar requirements. For example, an artificial delay between the direction pin and step pulse can be created by inverting the step pin.

This invert mask setting is a value which stores the axes to invert as bit flags. You really don't need to completely understand how it works. You simply need to enter the settings value for the axes you want to invert. For example, if you want to invert the X and Z axes, you'd send $2=5 to Grbl and the setting should now read $2=5 (step port invert mask:00000101).

Setting Value	Mask	Invert X	Invert Y	Invert Z
0	00000000	N	N	N
1	00000001	Y	N	N
2	00000010	N	Y	N
3	00000011	Y	Y	N
4	00000100	N	N	Y
5	00000101	Y	N	Y
6	00000110	N	Y	Y
7	00000111	Y	Y	Y

$3 – Direction port invert, mask

This setting inverts the direction signal for each axis. By default, Grbl assumes that the axes move in a positive direction when the direction pin signal is low, and a negative direction when the pin is high. Often, axes don't move this way with some machines. This setting will invert the direction pin signal for those axes that move the opposite way.

This invert mask setting works exactly like the step port invert mask and stores which axes to invert as bit flags. To configure this setting, you simply need to send the value for the axes you want to invert. Use the table above. For example, if want to invert the Y axis direction only, you'd send $3=2 to Grbl and the setting should now read $3=2 (dir port invert mask:00000010)

$4 - Step enable invert, boolean

By default, the stepper enable pin is high to disable and low to enable. If your setup needs the opposite, just invert the stepper enable pin by typing $4=1. Disable with $4=0. (May need a power cycle to load the change.)

$5 - Limit pins invert, boolean

By default, the limit pins are held normally-high with the Arduino's internal pull-up resistor. When a limit pin is low, Grbl interprets this as triggered. For the opposite behavior, just invert the limit pins by typing $5=1. Disable with $5=0. You may need a power cycle to load the change.

NOTE: For more advanced usage, the internal pull-up resistor on the limit pins may be disabled in config.h.

$6 - Probe pin invert, boolean

By default, the probe pin is held normally-high with the Arduino's internal pull-up resistor. When the probe pin is low, Grbl interprets this as triggered. For the opposite behavior, just invert the probe pin by typing $6=1. Disable with $6=0. You may need a power cycle to load the change.

$10 - Status report, mask

This setting determines what Grbl real-time data it reports back to the user when a '?' status report is sent. This data includes current run state, real-time position, real-time feed rate, pin states, current override values, buffer states, and the g-code line number currently executing (if enabled through compile-time options).

By default, the new report implementation in Grbl v1.1+ will include just about everything in the standard status report. A lot of the data is hidden and will appear only if it changes. This increases efficiency dramatically over of the old report style and allows you to get faster updates and still get more data about your machine. The interface documentation outlines how it works and most of it applies only to GUI developers or the curious.

To keep things simple and consistent, Grbl v1.1 has only two reporting options. These are primarily here just for users and developers to help set things up.

- Position type may be specified to show either machine position (MPos:) or work position (WPos:), but no longer both at the same time. Enabling work position is useful in certain scenarios when Grbl is being directly interacted with through a serial terminal, but *machine position reporting should be used by default.*
- Usage data of Grbl's planner and serial RX buffers may be enabled. This shows the number of blocks or bytes available in the respective buffers. This is generally used to helps determine how Grbl is performing when testing out a streaming interface. *This should be disabled by default.*

Use the table below enables and disable reporting options. Simply add the values listed of what you'd like to enable, then save it by sending Grbl your setting value. For example, the default report with machine position and no buffer data reports setting is $10=1. If work position and buffer data are desired, the setting will be $10=2.

Report Type	Value	Description
Position Type	0	Enable WPos: Disable MPos:.
Position Type	1	Enable MPos:. Disable WPos:.
Buffer Data	2	Enabled Buf: field appears with planner and serial RX available buffer.

$11 - Junction deviation, mm

Junction deviation is used by the acceleration manager to determine how fast it can move through line segment junctions of a G-code program path. For example, if the G-code path has a sharp 10 degree turn coming up and the machine is moving at full speed, this setting helps determine how much the machine needs to slow down to safely go through the corner without losing steps.

How we calculate it is a bit complicated, but, in general, higher values gives faster motion through corners, while increasing the risk of losing steps and positioning. Lower values makes the acceleration manager more careful and will lead to careful and slower cornering. So if you run into problems where your machine tries to take a corner too fast, *decrease* this value to make it slow down when entering corners. If you want your machine to move faster through junctions, *increase* this value to speed it up. For curious people, hit this link to read about Grbl's cornering algorithm, which accounts for both velocity and junction angle with a very simple, efficient, and robust method.

$12 – Arc tolerance, mm

Grbl renders G2/G3 circles, arcs, and helices by subdividing them into teeny tiny lines, such that the arc tracing accuracy is never below this value. You will probably never need to adjust this setting, since 0.002mm is well below the accuracy of most all CNC machines. But if you find that your circles are too crude or arc tracing is performing slowly, adjust this setting. Lower values give higher precision but may lead to performance issues by overloading Grbl with too many tiny lines. Alternately, higher values traces to a lower precision, but can speed up arc performance since Grbl has fewer lines to deal with.

For the curious, arc tolerance is defined as the maximum perpendicular distance from a line segment with its end points lying on the arc, aka a chord. With some basic geometry, we solve for the length of the line segments to trace the arc that satisfies this setting. Modeling arcs in this way is great, because the arc line segments automatically adjust and scale with length to ensure optimum arc tracing performance, while never losing accuracy.

$13 - Report inches, boolean

Grbl has a real-time positioning reporting feature to provide a user feedback on where the machine is exactly at that time, as well as, parameters for coordinate offsets and probing. By default, it is set to report in mm, but by sending a $13=1 command, you send this boolean flag to true and these reporting features will now report in inches. $13=0 to set back to mm.

$20 - Soft limits, boolean – modified by cnc4fun

Soft limits is a safety feature to help prevent your machine from traveling too far and beyond the limits of travel, crashing or breaking something expensive. It works by knowing the maximum travel limits for each axis and where Grbl is in machine coordinates. Whenever a new G-code motion is sent to Grbl, it checks whether or not you accidentally have exceeded your machine space. If you do, Grbl will issue an immediate feed hold wherever it is, shutdown the spindle and coolant, and then set the system alarm indicating the problem. Machine position will be retained afterwards, since it's not due to an immediate forced stop like hard limits.

NOTE: Soft limits requires homing to be enabled and accurate axis maximum travel settings, because Grbl needs to know where it is. $20=1 to enable, and $20=0 to disable.

CNC4FUN modifications:

Soft limits behaviour has been modified and is now working as follows: The original code didn't take into account the pull-off limit parameter which is now correctly subtracted.

After booting, soft limits are temporarily disabled anyway until the HOMING procedure has been performed, allowing you to move the axes in any direction without restrictions.

This is useful in many situations: for example, to quickly move the axes near the limit switch at power-up before the HOME procedure has been performed or when the board is used for laser applications.

The soft limits are then engaged independently (axis by axis) during the homing procedure. This behaviour is useful for example in laser applications, where you may not want to move the Z axis while homing ($59 = 3) but still want to be able to move the Z axis without any restrictions on soft limits. In this case, after homing, the X and Y axes are subject to soft limit checking, while the Z axis is still free to move in any directions without any checks.

PLEASE NOTE: if an axis homing is disabled via the $59 homing mask, its position value is assumed to be 0 (zero) at boot time.

$21 - Hard limits, boolean

Hard limit work basically the same as soft limits, but use physical switches instead. Basically you wire up some switches (mechanical, magnetic, or optical) near the end of travel of each axes, or where ever you feel that there might be trouble if your program moves too far to where it shouldn't. When the switch triggers, it will immediately halt all motion, shutdown the coolant and spindle (if connected), and go into alarm mode, which forces you to check your machine and reset everything.

To use hard limits with Grbl, the limit pins are held high with an internal pull-up resistor, so all you have to do is wire in a normally-open switch with the pin and ground and enable hard limits with $21=1. (Disable with $21=0.) We strongly advise taking electric interference prevention measures. If you want a limit for both ends of travel of one axes, just wire in two switches in parallel with the pin and ground, so if either one of them trips, it triggers the hard limit.

Keep in mind, that a hard limit event is considered to be critical event, where steppers immediately stop and will have likely have lost steps. Grbl doesn't have any feedback on position, so it can't guarantee it has any idea where it is. So, if a hard limit is triggered, Grbl will go into an infinite loop ALARM mode, giving you a chance to check your machine and forcing you to reset Grbl. Remember it's a purely a safety feature.

$22 - Homing cycle, boolean

Ahh, homing. For those just initiated into CNC, the homing cycle is used to accurately and precisely locate a known and consistent position on a machine every time you start up your Grbl between sessions. In other words, you know exactly where you are at any given time, every time. Say you start machining something or are about to start the next step in a job and the power goes out, you re-start Grbl and Grbl has no idea where it is due to steppers being open-loop control. You're left with the task of figuring out where you are. If you have homing, you always have the machine zero reference point to locate from, so all you have to do is run the homing cycle and resume where you left off.

To set up the homing cycle for Grbl, you need to have limit switches in a fixed position that won't get bumped or moved, or else your reference point gets messed up. Usually they are setup in the farthest point in +x, +y, +z of each axes. Wire your limit switches in with the limit pins, add a recommended RC-filter to help reduce electrical noise, and enable homing. If you're curious, you can use your limit switches for both hard limits AND homing. They play nice with each other.

Prior to trying the homing cycle for the first time, make sure you have setup everything correctly, otherwise homing may behave strangely. First, ensure your machine axes are moving in the correct directions per Cartesian coordinates (right-hand rule). If not, fix it with the $3 direction invert setting. Second, ensure your limit switch pins are not showing as 'triggered' in Grbl's status reports. If are, check your wiring and settings. Finally, ensure your $13x max travel settings are somewhat accurate (within 20%), because Grbl uses these values to determine how far it should search for the homing switches.

By default, Grbl's homing cycle moves the Z-axis positive first to clear the workspace and then moves both the X and Y-axes at the same time in the positive direction. To set up how your homing cycle behaves, there are more Grbl settings down the page describing what they do (and compile-time options as well.)

Also, one more thing to note, when homing is enabled. Grbl will lock out all G-code commands until you perform a homing cycle. Meaning no axes motions, unless the lock is disabled ($X) but more on that later. Most, if not all CNC controllers, do something similar, as it is mostly a safety feature to prevent users from making a positioning mistake, which is very easy to do and be saddened when a mistake ruins a part. If you find this annoying or find any weird bugs, please let us know and we'll try to work on it so everyone is happy. :)

NOTE: Check out config.h for more homing options for advanced users. You can disable the homing lockout at startup, configure which axes move first during a homing cycle and in what order, and more.

$23 - Homing dir invert, mask

By default, Grbl assumes your homing limit switches are in the positive direction, first moving the z-axis positive, then the x-y axes positive before trying to precisely locate machine zero by going back and forth slowly around the switch. If your machine has a limit switch in the negative direction, the homing direction mask can invert the axes' direction. It works just like the step port invert and direction port invert masks, where all you have to do is send the value in the table to indicate what axes you want to invert and search for in the opposite direction.

$24 - Homing feed, mm/min

The homing cycle first searches for the limit switches at a higher seek rate, and after it finds them, it moves at a slower feed rate to home into the precise location of machine zero. Homing feed rate is that slower feed rate. Set this to whatever rate value that provides repeatable and precise machine zero locating.

$25 - Homing seek, mm/min

Homing seek rate is the homing cycle search rate, or the rate at which it first tries to find the limit switches. Adjust to whatever rate gets to the limit switches in a short enough time without crashing into your limit switches if they come in too fast.

$26 - Homing debounce, milliseconds

Whenever a switch triggers, some of them can have electrical/mechanical noise that actually 'bounce' the signal high and low for a few milliseconds before settling in. To solve this, you need to debounce the signal, either by hardware with some kind of signal conditioner or by software with a short delay to let the signal finish bouncing. Grbl performs a short delay, only homing when locating machine zero. Set this delay value to whatever your switch needs to get repeatable homing. In most cases, 5-25 milliseconds is fine.

$27 - Homing pull-off, mm

To play nice with the hard limits feature, where homing can share the same limit switches, the homing cycle will move off all of the limit switches by this pull-off travel after it completes. In other words, it helps to prevent accidental triggering of the hard limit after a homing cycle. Make sure this value is large enough to clear the limit switch. If not, Grbl will throw an alarm error for failing to clear it.

$30 - Max spindle speed, RPM

This sets the spindle speed for the maximum 5V PWM pin output. For example, if you want to set 10000rpm at 5V, program $30=10000. For 255rpm at 5V, program $30=255. If a program tries to set a higher spindle RPM greater than the $30 max spindle speed, Grbl will just output the max 5V, since it can't go any faster. By default, Grbl linearly relates the max-min RPMs to 5V-0.02V PWM pin output in 255 equally spaced increments. When the PWM pin reads 0V, this indicates spindle disabled. Note that there are additional configuration options are available in config.h to tweak how this operates.

$31 - Min spindle speed, RPM

This sets the spindle speed for the minimum 0.02V PWM pin output (0V is disabled). Lower RPM values are accepted by Grbl but the PWM output will not go below 0.02V, except when RPM is zero. If zero, the spindle is disabled and PWM output is 0V.

$32 - Laser mode, boolean

When enabled, Grbl will move continuously through consecutive G1, G2, or G3 motion commands when programmed with a S spindle speed (laser power). The spindle PWM pin will be updated instantaneously through each motion without stopping. Please read the GRBL laser documentation and your laser device documentation prior to using this mode. Lasers are very dangerous. They can instantly damage your vision permanently and cause fires. Grbl does not assume any responsibility for any issues the firmware may cause, as defined by its GPL license.

When disabled, Grbl will operate as it always has, stopping motion with every S spindle speed command. This is the default operation of a milling machine to allow a pause to let the spindle change speeds.

$33 – MIST-ON only when Z axis is down – added by cnc4fun

If set to 1, the MIST relay is active only when the Z axis Is down and working. This function can be useful to save refrigeration liquid.

$50 – Expansion board, boolean – added by cnc4fun

If set to 1, it means that the CNC4FUN expansion board is installed.

$51 – Pendant enabled, boolean – added by cnc4fun

If set to 1, it means that the CNC4FUN pendant keyboard is installed.

$52 – Spindle off at 0 speed, boolean – added by cnc4fun

If set to 1, Grbl turns off the spindle motor relay when the spindle speed is set to 0.

$53 – Parking axis target, mm – added by cnc4fun (UNDER DEVELOPMENT)

Sets the parking target in millimeters as machine coordinates [-max_travel,0].

$54 – Parking rate, mm/min – added by cnc4fun (UNDER DEVELOPMENT)

Sets the parking fast rate after pullout in mm/min.

$55 – Parking pullout rate, mm/min – added by cnc4fun (UNDER DEVELOPMENT)

Sets the pullout/plunge slow feed rate in mm/min.

$56 – Parking pullout increment, mm/min – added by cnc4fun (UNDER DEVELOPMENT)

Spindle pull-out and plunge distance in mm. Incremental distance. Must be a positive value or a value equal to zero.

$57 – Spindle PWM, Hz – added by cnc4fun

Sets the PWM frequency of the spindle speed control. 15/20 KHz can be okay to control a spindle motor in DC, but more often than not lasers for non-professional use can’t work at frequencies that are higher than 500/1000 Hz. This parameter allows to set a specific frequency. The change takes effects after a reboot.

$58 – Servo wakeup time, ms – added by cnc4fun

Depending on the specific motors and servo drivers that are present on the user’s CNC, before any movement can be executed by the machine you may need to wait some time from power-up until the servos are ready. This command allows you to set this waiting time in milliseconds.

$59 – Enable, mask (XYZ) – added by cnc4fun

Depending on whether you have a CNC or a laser cutter, you may want to disable the Z axis homing function. The original GRBL code only provides a global ON/OFF function, either enabling or disabling all the homing axes at the same time ($21).

For compatibility reasons, the $21 parameter is still present, but the $59 parameter allows you to enable/disable each single axis separately.

Setting Value	Mask	Home X	Home Y	Home Z
0	00000000	N	N	N
1	00000001	Y	N	N
2	00000010	N	Y	N
3	00000011	Y	Y	N
4	00000100	N	N	Y
5	00000101	Y	N	Y
6	00000110	N	Y	Y
7	00000111	Y	Y	Y

$60 to $65 ; $70 to $75 – Virtual stepped knob parameters – added by cnc4fun

The two potentiometers on the CNC4FUN board are ‘virtualized’ as stepped knobs. In order to correctly set up the machine, you will need to enter some calibration values that will enable the virtual knobs to work properly.

When you work on a CNC, you have a nominal value for the spindle speed and the feed rate. You can override the nominal value by using the correction knobs. By definition, the central position of the knob represents the nominal value (1x). Rotating the knob to the left means reducing the value, while rotating it to the right means increasing the value.

Depending on your needs (laser mode, CNC mode, spindle requirements, etc.) you may need to limit the range of correction of the nominal value. In order to do this, some calibrations and configurations are needed. The following commands will allow you to do this.

The knob virtualization process is done in three steps:

1. data acquisition and filtering from ADC,
2. linearization,
3. stepping and scaling of the values.

Step 1: enter the minimum, middle and maximum value. This means entering the values you read from the ADC (0-1023) when the potentiometer is in the leftmost position, in the central position, and finally in the rightmost position ($60, $61, $62, $70, $71, $72).

Step 2: the system will linearize the value based on the data entered by the user.

Step 3: set the number of positions of the virtual knob you want to have ($65, $75). Then, set the range of allowed corrections ($63, $64, $73, $74).

Obtaining the current values:

The two commands @69 and @79 allow you to read the current ADC and scaled values. For example:

```
@69 = ADC Readings now: - ADC=339 Scaled=45% Knob=1.0
```

You can use the current ADC value to properly set the relevant parameters.

NOTE:

For technical reasons, there are two ways of setting the ADC values:

1. reading the values via the @69 and @79 commands, or
2. automatically acquiring the current ADC value (see below).

$60 – Spindle knob min value – added by cnc4fun

Sets the ADC value when the potentiometer is in the leftmost position.

To automatically read the current ADC value, use <$60 = -1> as a value.

$61 – Spindle knob mid value – added by cnc4fun

Sets the ADC value when the potentiometer is in the central position.

To automatically read the current ADC value, use <$61 = -1> as a value.

$62 – Spindle knob min value – added by cnc4fun

Sets the ADC value when the potentiometer is in the rightmost position.

To automatically read the current ADC value, use <$62 = -1> as a value.

$63 – Spindle knob min multiplier – added by cnc4fun

Sets the maximum attenuation limit for the virtual knob.

$64 – Spindle knob max multiplier – added by cnc4fun

Sets the maximum gain limit for the virtual knob.

$65 – Spindle knob steps – added by cnc4fun

Sets the number of virtual knob steps.

$66 – Spindle spin-up time – added by cnc4fun

Sets the number of milliseconds to wait from the M3 command (power supply relay ON) to the next G-code command execution (useful to provide some time for the laser power-supply stabilization).

@69 – Spindle ADC reading – added by cnc4fun

This command returns the current ADC reading and the knob status.

See the example below:

```
@69 = ADC Readings now: - ADC=859 Scaled=100% Knob=3.0
```

$70 – Feed knob min value – added by cnc4fun

Sets the ADC value when the potentiometer is in the leftmost position.

To automatically read the current ADC value, use <$70 = -1> as a value.

$71 – Feed knob mid value – added by cnc4fun

Sets the ADC value when the potentiometer is in the central position.

To automatically read the current ADC value, use <$71 = -1> as a value.

$72 – Feed knob min value – added by cnc4fun

Sets the ADC value when the potentiometer is in the rightmost position.

To automatically read the current ADC value, use <$72 = -1> as a value.

$73 – Feed knob min multiplier – added by cnc4fun

Sets the maximum attenuation limit for the virtual knob.

$74 – Feed knob max multiplier – added by cnc4fun

Sets the maximum gain limit for the virtual knob.

$75 – Feed knob steps – added by cnc4fun

Sets the number of virtual knob steps.

@79 – Feed ADC reading – added by cnc4fun

This command returns the current ADC reading and the knob status.

See the example below:

```
@79 = ADC Readings now: - ADC=859 Scaled=100% Knob=3.0
```

$76 – Feed knob hold at 0 speed – added by cnc4fun

If set to 1, put the CNC on hold when the knob is in the leftmost position.

$77 – Feed minimum speed – added by cnc4fun

Sets the minimum feed speed that the CNC is allowed to use.

$80 – Pendant fast mode timer, [X,Y] sec.– added by cnc4fun

Sets the time (in seconds) before the XY axes movement changes from slow-mode to fast-mode.

$81 – Pendant fast mode timer, [Z] sec.– added by cnc4fun

Sets the time (in seconds) before the Z axis movement changes from slow-mode to fast-mode.

$82 – Pendant jog feed sped, [X,Y] mm/min.– added by cnc4fun

Sets the feed rate of the XY axes movement.

$83 – Pendant jog feed sped, [Z] mm/min.– added by cnc4fun

Sets the feed rate of the Z axis movement.

$84 – Pendant jog step, [X,Y,Z] mm – added by cnc4fun

On the pendant there are two “Jog” step pushbuttons. The combination of the two sets the axis movement (in millimeters) for each single step.

This parameter sets the step of XYZ axes movement when the pushbuttons are =00.

$85 – Pendant jog step, [X,Y,Z] mm – added by cnc4fun

On the pendant there are two “Jog” step pushbuttons. The combination of the two sets the axis movement (in millimeters) for each single step.

This parameter sets the step of XYZ axes movement when the pushbuttons are =01.

$86 – Pendant jog step, [X,Y,Z] mm – added by cnc4fun

On the pendant there are two “Jog” step pushbuttons. The combination of the two sets the axis movement (in millimeters) for each single step.

This parameter sets the step of XYZ axes movement when the pushbuttons are =10.

$87 – Pendant jog step, [X,Y,Z] mm – added by cnc4fun

On the pendant there are two “Jog” step pushbuttons. The combination of the two sets the axis movement (in millimeters) for each single step.

This parameter sets the step of XYZ axes movement when the pushbuttons are =11.

$88 –Swap mask for XYZ pushbuttons – added by cnc4fun

Accordind to your preferences, you may want to invert the action of the XYZ pushbuttons on the pendant of your CNC. This is the relevant swap mask.

Setting Value	Mask	Invert X	Invert Y	Invert Z
0	00000000	N	N	N
1	00000001	Y	N	N
2	00000010	N	Y	N
3	00000011	Y	Y	N
4	00000100	N	N	Y
5	00000101	Y	N	Y
6	00000110	N	Y	Y
7	00000111	Y	Y	Y

$89 – Pendant soft reset = REBOOT after xx Sec. – added by cnc4fun

The PENDANT soft-reset button will perform a hardware CPU REBOOT if pressed longer than the time specified by this parameter.

$90 – WiFi off – added by cnc4fun

Forces WiFi off. When set to 1, this command disables the wifi interface.

$91 – Serial port baud rate – added by cnc4fun

Selects the operational speed of the serial communication port. The supported baudrates are: 115200, 23400, 250000, 460800, 500000, 921600, 1000000

$100, $101 and $102 – [X,Y,Z] steps/mm

Grbl needs to know how far each step will take the tool in reality. To calculate steps/mm for an axis of your machine you need to know:

- The mm traveled per revolution of your stepper motor. This is dependent on your belt drive gears or lead screw pitch.
- The full steps per revolution of your steppers (typically 200)
- The microsteps per step of your controller (typically 1, 2, 4, 8, or 16). *Tip: Using high microstep values (e.g., 16) can reduce your stepper motor torque, so use the lowest that gives you the desired axis resolution and comfortable running properties.*

The steps/mm can then be calculated like this: steps_per_mm = (steps_per_revolution*microsteps)/mm_per_rev

Compute this value for every axis and write these settings to Grbl.

$110, $111 and $112 – [X,Y,Z] Max rate, mm/min

This sets the maximum rate each axis can move. Whenever Grbl plans a move, it checks whether or not the move causes any one of these individual axes to exceed their max rate. If so, it'll slow down the motion to ensure none of the axes exceed their max rate limits. This means that each axis has its own independent speed, which is extremely useful for limiting the typically slower Z-axis.

The simplest way to determine these values is to test each axis one at a time by slowly increasing max rate settings and moving it. For example, to test the X-axis, send Grbl something like G0 X50 with enough travel distance so that the axis accelerates to its max speed. You'll know you've hit the max rate threshold when your steppers stall.

It'll make a bit of noise, but shouldn't hurt your motors. Enter a setting a 10-20% below this value, so you can account for wear, friction, and the mass of your workpiece/tool. Then, repeat for your other axes.

NOTE: This max rate setting also sets the G0 seek rates.

$120, $121, $122 – [X,Y,Z] Acceleration, mm/sec^2

This sets the axes acceleration parameters in mm/second/second. Simplistically, a lower value makes Grbl ease slower into motion, while a higher value yields tighter moves and reaches the desired feed rates much quicker. Much like the max rate setting, each axis has its own acceleration value and are independent of each other. This means that a multi-axis motion will only accelerate as quickly as the lowest contributing axis can.

Again, like the max rate setting, the simplest way to determine the values for this setting is to individually test each axis with slowly increasing values until the motor stalls. Then finalize your acceleration setting with a value 10-20% below this absolute max value. This should account for wear, friction, and mass inertia. We highly recommend that you dry test some G-code programs with your new settings before committing to them. Sometimes the loading on your machine is different when moving in all axes together.

$130, $131, $132 – [X,Y,Z] Max travel, mm

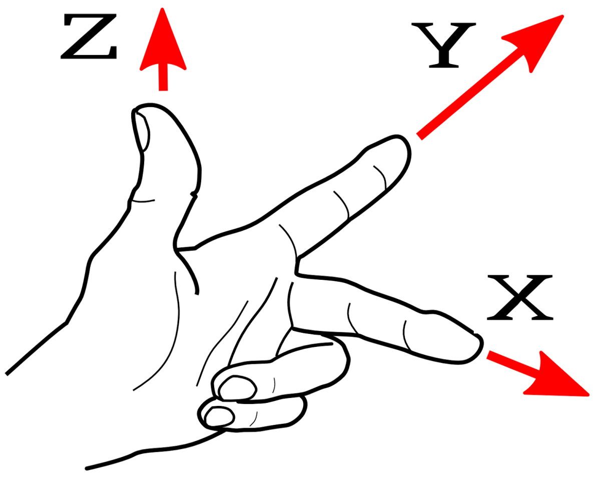
This sets the maximum travel from end to end for each axis in mm. This is only useful if you have soft limits (and homing) enabled, as this is only used by Grbl's soft limit feature to check if you have exceeded your machine limits with a motion command.

Quick Guide to Setting Up Your Machine for the First Time

Grbl's default configuration is intentionally very generic to help ensure users can see successful motion without having to tweak settings. Generally, the first thing you'll want to do is get your stepper motors running, usually without it connected to the CNC. Wire Grbl to your stepper drivers and stepper motors according to your manufacturer guidelines. Connect to Grbl through a serial terminal or one of many Grbl GUIs. Send some G1 or G0 commands to Grbl. You should see your stepper motor rotating. If you are having trouble with your stepper motors, try the following:

- Ensure everything is wired and powered correctly per your stepper driver manufacturer guidelines.
- If your steppers are mounted in your CNC already, ensure your axes move freely and don't obviously bind. If you can't easily tell, try removing your steppers and check if they run under no load.
- Ensure your stepper motors and axes linear mechanisms are all tight and secure. Small set screws on drivetrain components becoming loose is a very common problem. Re-tighten and try applying some non-permenant thread locker (Loctite blue) if it continually loosens.
- For more difficult issues, try the process of elimination to quickly isolate the problem. Start by disconnecting everything from the Arduino. Test if Grbl is operating ok by itself. Then, add one thing at a time and test.
- If your steppers are powered and making a grinding noise when trying to move, try lowering the '$' acceleration and max rate settings. This sound is a sign that your steppers is losing steps and not able to keep up due too much torque load or going too fast.
- Grbl's default step pulse settings cover the vast majority of stepper drivers on the market. While very uncommon, check these settings if you are still experiencing problems or have a unusual setup.

Next, you will need to make sure your machine is moving in the correct directions according to a Cartesian(XYZ) coordinate frame and satisfies the rule, as shown:



Mount your stepper motors into your CNC, if you haven't already done so. Send Grbl some motion commands, such as G91 G0 X1 or G91 G0 X-1, which will move the x-axis +1mm and -1mm, respectively. Check all axes. If an axis is not moving correctly, alter the $3 direction port mask setting to invert the direction.

If you are unfamiliar with how coordinate frames are setup on CNC machines, see this great diagram by LinuxCNC. Just keep in mind that motions are *relative* to the tool. So on a typical CNC gantry router, the tool will move rather than the fixed table. If the x-axis is aligned positive to the right, a positive motion command will move the tool to the right. Whereas, a moving table with a fixed tool will move the table to the left for the same command, because the tool is moving to the right relative to the table.

Finally, tune your settings to get close to your desired or max performance. Start by ensuring your $100,$101, and $102 axes step/mm settings are correct for your setup. This is dependent on your stepper increments, micro steps on your driver, and mechanical parameters. There are multiple resources online to show you how to compute this for your particular machine, if your machine manufacturer has not supplied this for you. Tweak your $12x acceleration and $11x max rate settings to improve performance. Set to no greater than 80% of absolute max to account for inertia, cutting forces, and motor torque reductions with speed. Set your $13x max travel settings if you plan on using homing or soft limits. It's recommended to enter something approximately close to actual travel now to avoid problems in the future.

At this point, you're pretty much ready to get going! Grbl can now move your CNC machine and run g-code jobs. If you need to add more features, such as limit switches for homing or hard limits or spindle/laser control. There are other Wiki pages to help you that. Good luck and have fun!