

# COAST RUNNER OPERATOR'S MANUAL

An open source project by **COAST RUNNER, INC.**

0 - Welcome to Coast Runner!.....	1
1 - Safety.....	2
2 - Unboxing .....	3
2.1 - Packing List .....	3
3 - Physical Overview.....	4
4. - Operating Environment.....	5
6 - Attaching Work Piece .....	6
6.1 - Threading Bolts into T-Nuts .....	6
6.2 - Removing T-Nuts .....	8
7 - Installing Tool .....	9
8 - Installing Chip Cover .....	11
9 - Probing - Theory of Operation.....	12
9.1 - Probing Anodized Parts .....	14
9.2 - Probing Non-Conductive Parts .....	14
10 - FAQ/Troubleshooting .....	15
11 - Maintenance .....	18
11.1 - Removing Chip Accumulation .....	18
11.2 - Protection from Rust .....	18
12 - Leveling X Table.....	19
12.1 - \$L Theory of Operation .....	19
12.2 - Manually Recalibrating 'Level' .....	19
12.3 - \$LS Theory of Operation .....	19
13 - Updating Firmware .....	20
13.1 - Automatic Updates .....	20
13.2 - Manual Updates .....	20
13.3 - Writing Custom Firmware .....	20
14 - CRWrite - Privacy .....	21
14.1 - CRWrite Log File .....	21
14.2 - Other Information CRWrite Stores Locally .....	21
14.3 - Other Information CRWrite sends over the Internet .....	21
15 - RMA Policy .....	22
16 - Technical Overview .....	23
17 - Connecting to Coast Runner .....	24
17.1 - Connecting to Coast Runner - Run .crproj File .....	24
17.2 - Connecting to Coast Runner - CRWrite Manual Operations window .....	24
17.3 - Connecting to Coast Runner - 3rd Party Grbl-Compatible Controller .....	26
17.4.1 - Connecting to Coast Runner - Serial Terminal (Arduino) .....	26
17.4.2 - Connecting to Coast Runner - Serial Terminal (PTY) .....	28
17.5 - Connecting to Coast Runner - Conclusion .....	28
18 - Interacting with Grbl - Hands on Example .....	29
18.1 - So What is Grbl, Exactly? .....	29
18.2 - Interacting with Grbl - Getting Started .....	29
18.2.1 - Querying Status/Settings .....	30
18.3 - Manually Moving Axes .....	31

18.3.1 - Right Hand Rule	35
18.3.2 - Soft Limits	36
18.3.3 - Tool Install Coordinates	37
18.4 - Modal Parameters	37
18.5.1 - Relative Motion	39
18.5.2 - Absolute Motion	39
18.6 - Controlling the Spindle	40
18.7 - Probing with G-code	41
18.7.1 - Common Probe Failures	42
18.7.2 - Rotating Spindle While Probing	42
18.7.3 - Probing Example	43
18.8 - Work Coordinate System (WCS) Offsets	44
18.8.1 - Setting WCS Offsets - Absolute (L2)	45
18.8.2 - Setting WCS Offsets - Relative (L20)	46
18.8.3 - Accounting for Tool Diameter	46
18.8.4 - WCS Offsets Are Persistent	47
18.8.5 - The WCS Clear Popup	47
19 - Creating .crproj Files.....	49
19.1 - Creating the manifest.yml File	49
19.1.1 - Example manifest.yml File	51
19.2 - Advanced manifest.yml Features	51
19.2.1 - Using Markdown Formatting	51
19.2.2 - Hiding the WCS Clear Popup	52
19.2.3 - Using Submanifest Files	52
19.2.4 - Enforcing a Minimum Software Version	53
19.2.5 - Enforcing a Minimum Firmware Version	54
19.2.6 - Enforcing a WCS Offset for a File	54
19.2.7 - Making a Step Unskippable	54
19.2.8 - Jumps	55
19.2.8 - Popups	55
19.2.9 - Popups and Jumps	56
19.3 - Including Additional Files for a User to Extract	56
19.4 - Creating G-Code	57
19.5 - Designing Custom Jigs	58
19.5 - X Table T-Slot Geometry	59
Appendix A: Supported G-Code Commands.....	60
Appendix B: Grbl-Specific Commands.....	65
Grbl-Specific Commands - Response Syntax	66
Appendix C: Voltage Selector Switch .....	68
Powering Coast Runner with a 240 volt AC line input	68
Powering Coast Runner with a 120 volt AC line input	68
Appendix E: Errors & Alarms.....	69
Appendix G: Grbl Settings (control parameters).....	71
Appendix H: Differences Between Grbl & Our Fork.....	73
Appendix K: Disassembling Coast Runner .....	74

Reassembling Coast Runner .....	80
Appendix L: Replacing Spindle Belt .....	81
Appendix N: Specifications.....	82
Appendix P: Axis Labels .....	83

**Version: 2024AUG30**



# 0 - Welcome to Coast Runner!

Thank you for purchasing a Coast Runner desktop CNC!

We believe Coast Runner is the most capable machine in its class, particularly since it includes many high end features not typically available on desktop CNC machines: automatic table leveling; electronic part probing; a custom-designed low runout spindle coupled to a closed loop VFD spindle controller; precision optical limit switches; and low backlash precision linear ballscrews. While Coast Runner represents a huge technological improvement over other desktop CNC mills, it is still not a high-end CNC mill and so comes with certain limitations. We believe that Coast Runner strikes a balance between affordability and capability. For the money, we believe Coast Runner's capabilities cannot be matched.

We have spent ten years meticulously designing the CNC machine that now sits before you. We sincerely hope that Coast Runner meets your prototyping and manufacturing needs. If not, please don't hesitate to contact us: [support@coastrunner.net](mailto:support@coastrunner.net).

We cannot overstate our gratitude that you have chosen to purchase a Coast Runner. Thank you!

---

## "Do I need to read this manual?"

Yes. Please read to at least the following page:

Experience Level	Read Manual Through Page:	What You'll Learn
"I'm a first time user."	6	Basic information and setup
"I've used Coast Runner before."	24	How to maintain and troubleshoot common problems you may encounter during normal use.
"I'm ready to learn how to design my own parts."	Read entire manual.	How to design new parts.

This manual includes solutions to the most common customer issues we've seen over the past five years. Even our most experienced CNC users should read this manual - or at least skim it over - prior to using Coast Runner.

# 1 - Safety

Ignorance is dangerous. Read and understand this manual prior to using Coast Runner.

The following safety alert words are used throughout this manual:

**Danger:** Indicates a hazardous situation that will result in death or serious injury.

**Warning:** Indicates a hazardous situation that could result in death or serious injury.

**Caution:** Indicates a hazardous situation that could result in minor or moderate injury.

**Note:** Indicates information considered important but not hazard related.

**Warning:** Crush hazard. Coast Runner's gantry and table can crush, pinch and tear body parts. Do not reach into Coast Runner except as indicated, AND only when the machine is at a complete stop.

**Warning:** Extremely sharp rotating cutter inside. Secure spindle with 12 mm wrench prior to servicing cutting tool, following the procedure outlined in this manual. The spindle will not spin when sufficient holding force is applied via the 12 mm wrench.

**Warning:** Never modify the included power cable in any way. Do NOT - under any circumstances - cut, remove or bypass the safety ground prong. Do NOT plug the cable into an outlet that is not properly grounded. Defeating the safety ground terminal could result in electric shock (e.g. caused by stray chips shorting mains to the metal enclosure).

**Caution:** Keep hands away from all pinch points and cutting surfaces whenever power is applied and the emergency stop switch is not engaged. Coast Runner is computer controlled and can start automatically.

**Caution:** Never use a dropped, visibly damaged, dull, or suspect cutting tool. Worn tools - particularly carbide tools - are brittle and could shatter while the tool is in motion. Discard all suspect tools.

**Caution:** Wear safety glasses when the spindle is rotating. Coast Runner's clear chip cover is not designed to contain an improperly secured work piece or shattered end mill.

**Caution:** Wear hearing protection when operating Coast Runner.

**Caution:** Do not operate Coast Runner with rings, watches, necklaces, loose clothing, or long hair down.

**Caution:** Waste chips produced during steel machining operations are razor sharp. Clean with vacuum.

**Note:** Coast Runner has a dedicated hardware emergency stop button. When engaged, all motion is immediately disabled (spindle and all three axes), and the connection to the host is disabled.

**Caution:** The host software (CRWrite) includes a virtual emergency stop button. Clicking this virtual button will only stop Coast Runner if CRWrite is connected to Coast Runner.

**Caution:** The spindle and stepper motors generate heat during use. Touching these parts could cause minor burn injuries, particularly immediately after machining parts.

**Caution:** Machined parts retain heat during the manufacturing process. Do not touch parts until cool.

**Note:** Coast Runner is not a consumer device. It is the user's responsibility to operate Coast Runner per OSHA 1910.212 - Milling Machine, ANSI B11.8-1983, and OSHA 3067, as amended.

## 2 - Unboxing

Coast Runner ships upside down. To remove Coast Runner from the shipping container and prepare for use:

A: Remove the cardboard accessory box.

B: Remove QTY4 foam corners.

C: Lift Coast Runner out of box, using built-in handles on either side.

**Note:** The emergency stop button is NOT a handle.

D: Place Coast Runner (still upside down) on a level work surface.

E: Remove shipping tape that secures the clear chip cover to the enclosure.

**Note:** Removing the tape is more difficult once the machine is right-side-up.

F: Tip Coast Runner forward and gently rotate until the large opening is on the bottom.

**Note:** Do not rest Coast Runner on the emergency stop button (located on Coast Runner's left side).

G: Remove the large shipping foam insert from the cardboard box and place on a level work surface.

**Note:** Keep the box and foam for future transportation. Protecting a 43 pound precision milling machine during shipping is difficult without specific packaging; a collapsed box takes little space. Do not ship Coast Runner without OEM packaging! Replacement packaging can be purchased if needed.

H: Place Coast Runner into the cutout in the large shipping foam insert. This helps contain chips.

**Caution:** Without the foam insert, Coast Runner can 'walk' on low friction surfaces. If placed directly onto a low friction surface (e.g. stainless steel, glass, etc), Coast Runner could fall to the floor below!

J: Connect Coast Runner's included power cable to a properly grounded 120 volt power outlet.

**Warning:** See Appendix C to operate Coast Runner at 240 volt mains (using a 3rd-party power cable).

**Warning:** To ensure continued safe operation, Coast Runner MUST connect to earth ground via a three-pronged power cord connected to a properly grounded outlet. Defeating the ground protection prong on the included power cable poses an electrocution hazard (e.g. a stray chip shorted from mains to the metal enclosure). Coast Runner will still work with the ground prong removed, but could electrocute you. DO NOT DEFEAT GROUND! COAST RUNNER MUST BE GROUNDED!

K: Connect Coast Runner's USB cable directly to the host computer.

**Note:** Coast Runner requires a low-latency connection to the host computer. Coast Runner should operate properly when plugged into a powered USB hub, but we recommend connecting directly to the host computer.

L: Rotate the red emergency stop button clockwise until it pops out.

**Note:** Coast Runner cannot communicate with the host computer when the emergency stop button is engaged.

### 2.1 - Packing List

The following items are included in the cardboard accessory box:

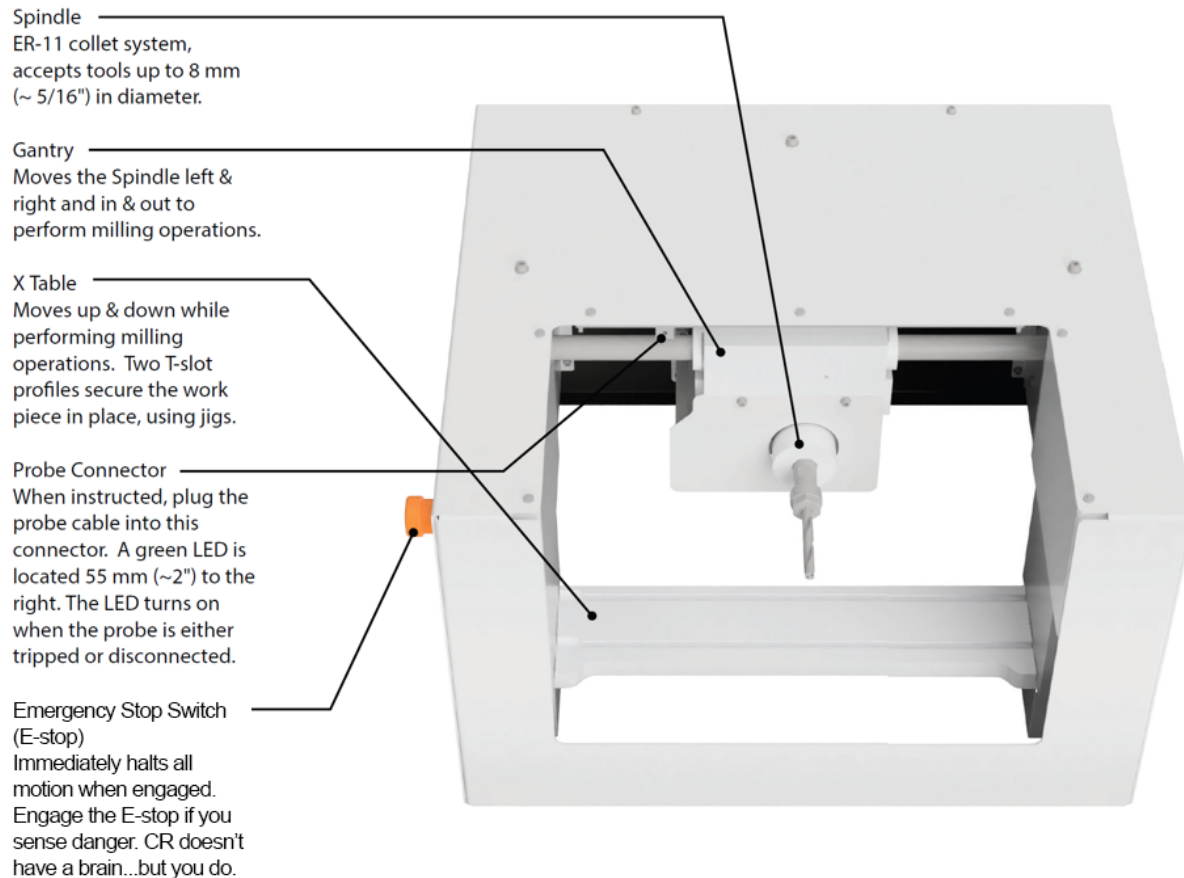
Part	Quantity
17 mm open-ended wrench	1
12 mm open-ended wrench	1
ER11 collet nut	1
USB A-B cable (2 meter)	1
NEMA 5-15/IEC C13 power cable	1
USB Flash Drive	1
Red Probe Cable	1
3 mm Allen Wrench	1



Milling parts requires tools and fixturing, such as the Coast Runner Universal Clamp, which are not included and are sold separately.

### 3 - Physical Overview

Take a moment to identify the following features on your Coast Runner:



To **engage** the emergency stop, push down on the red button until it clicks. Once engaged:

- all linear motion is immediately disabled, and;
- the spindle and stepper motors are abruptly powered down, and;
- the motion planner is placed into reset, and;
- Coast Runner will not process or respond to ANY command (they are discarded), and;
- The LED work lights & left cooling fan remain on.

Engaging the E-stop does not harm Coast Runner, but will cancel the current operation.

**Note:** If Coast Runner won't respond to commands, check the E-stop! CRWrite cannot talk to Coast Runner when engaged. We're going to keep reiterating this point throughout the manual... it's our #1 customer support issue.

To **disengage** the E-stop, rotate the red button clockwise until it pops out. Once disengaged, Coast Runner reinitializes to its default power-up settings; only persistent values are retained (see 10-FAQ Q19 for persistent list).

**Warning:** When the power & USB cables are connected - and the emergency stop is disengaged - software commands can cause the Gantry, X Table, and/or Spindle to move at any time. When these conditions are met, DO NOT reach into Coast Runner except as specifically indicated in this manual.

## 4. - Operating Environment

Operate Coast Runner in a shop environment conducive to loud noises and stray metal chips. Coast Runner's enclosure is designed to retain most chips produced during the milling process. However, some chips will leave the enclosure (primarily through the bottom opening).

Stray metal chips can damage electronic devices, including the host computer. Therefore, place Coast Runner:

- At least one meter away from electronic devices (the included USB cable is two meters long), and;
- Below electronic devices (e.g. by placing the computer on a book, block, etc).

Coast Runner's chip guards rely on gravity to prevent chips from entering sensitive components. Therefore, Coast Runner must operate in the upright position. Always vacuum Coast Runner prior to turning upside down.

**Do not use compressed air inside Coast Runner.** We absolutely cannot over-stress this point. Compressed air easily defeats Coast Runner's gravity-based chip guards. Chip accumulation behind these guards can seize the bearings and/or damage Coast Runner's electronics. Our RMA team can easily determine when a machine was (incorrectly) cleaned with compressed air (e.g. chips are visible behind nearly every guard).

**Note:** *Vacuuming is the **\*\*ONLY\*\*** approved method to remove chips from Coast Runner.*

---

**NEW USERS CAN STOP READING THE MANUAL HERE.**

## 6 - Attaching Work Piece

This section provides supplemental information and is a general overview on how to mechanically secure a fixture or jig to Coast Runner's X Table. We've found that users unfamiliar with T-Slot systems are more likely to succeed after reading this supplemental section. No action is required while reading this section; just learn the concept.

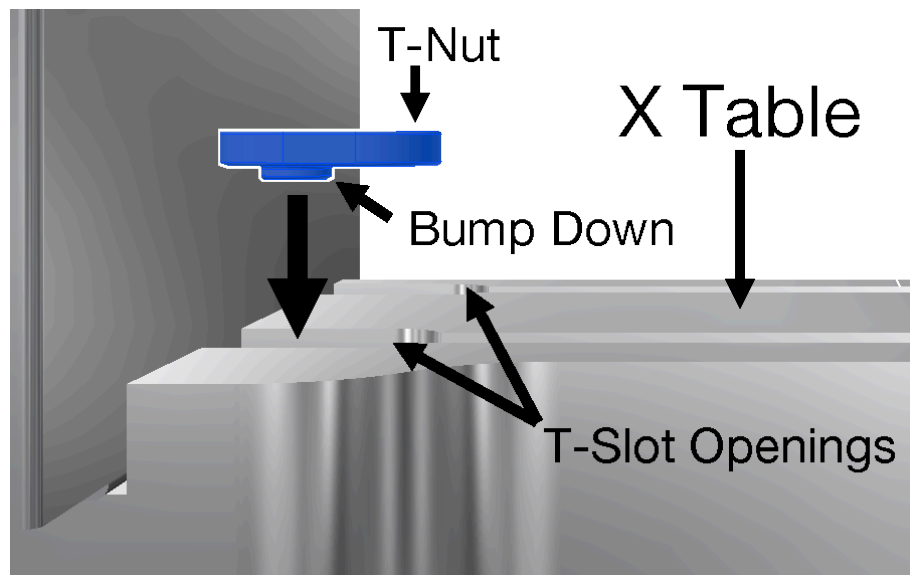
**Note:** CRWrite will tell you when to perform the steps outlined below.

After reading this section, you should be able to install a T-Nut into the T-Slot, then securely mount the work piece to the X Table. This section will seem trivial if you've used a T-Slot system before. We still recommend reading it.

**Note:** Our customers tend to not require additional help when securing the fixture to the work piece. The steps below assume the fixture and work piece are attached, but not mounted to the X Table.

There are two T-Slot profiles running lengthwise across the X Table. These profiles are used to securely fasten fixtures to Coast Runner. The left side of each profile has an opening, which allows T-Nuts to slide into the T-Slot profiles. When ready to mount a fixture, insert T-nuts into these openings, making sure that the 1 mm boss ('bump') faces down:

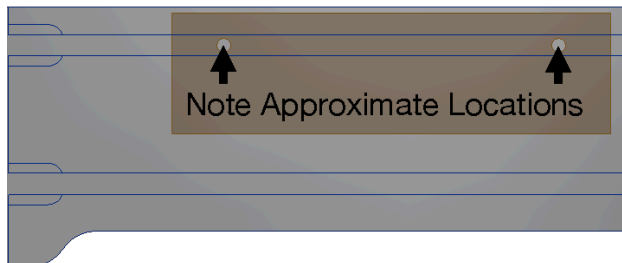
**Note:** The bump on each T-nut must face down!!! Installing T-Nuts with the bump up concentrates the clamping force onto a smaller surface area, which can damage the T-Slot and/or fixtures.



Perspective view of X Table, showing T-Nut insertion into lower T-Slot profile

### 6.1 - Threading Bolts into T-Nuts

Follow these steps to fasten a fixture to the X Table:



A1: Place the fixture onto the X Table.

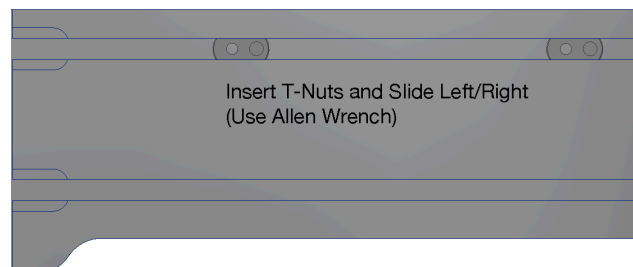
A2: Locate each fixture hole's approximate location.

**Note:** The transparent fixture shown to the left has two mounting holes in the top profile, and no mounting holes in the bottom profile.

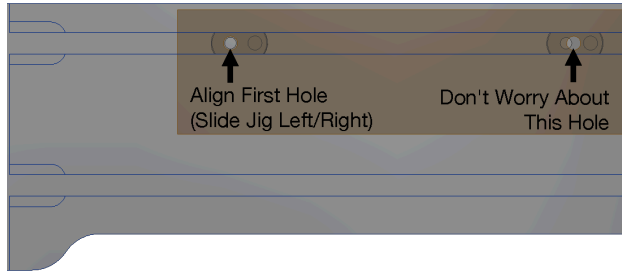
B: Remove the fixture. Insert a T-Nut into the profile, then slide to approximate fixture hole position.

**Note:** Slide the T-Nut with the Allen wrench.

**Note:** Repeat this step for each hole.



**Note:** Perfect hole alignment isn't required.



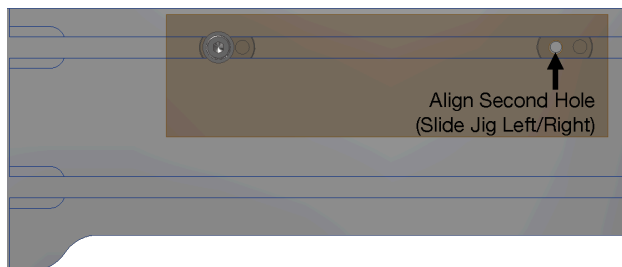
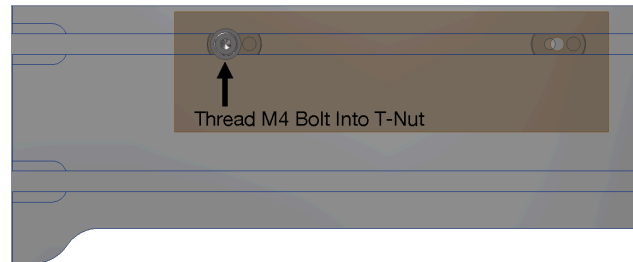
C1: Place the fixture onto the X Table. The boss on the fixture's bottom must insert into the profile\*.  
C2: Slide the fixture left/right to align at least one hole with the threads in its mating T-Nut.

**Note:** Don't worry about simultaneously aligning the other holes with their T-Nuts.

D: Place an M4 bolt into the hole and loosely thread into the T-Nut.

**Note:** Each fixture requires specific length bolts. Bolts that are too long prevent the fixture from clamping to the X Table. Bolts that are too short won't thread into the T-Nuts.

**Note:** If the bolt won't engage the T-Nut's threads, verify the fixture's bottom boss is recessed into the T-Slot profile\*.



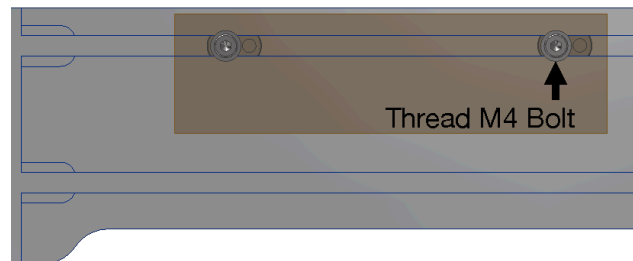
E: Slide the fixture left/right until the next hole is aligned with its mating T-Nut thread.

**Note:** If the fixture won't slide, loosen the previously installed bolt.

F: Loosely thread an M4 bolt into the T-Nut.

G: Repeat steps E & F for any remaining bolts.

H: Slide the fixture to the approximate probing/machining position.



J: Tighten all mounting bolts. The fixture should be firmly mounted to the X Table.

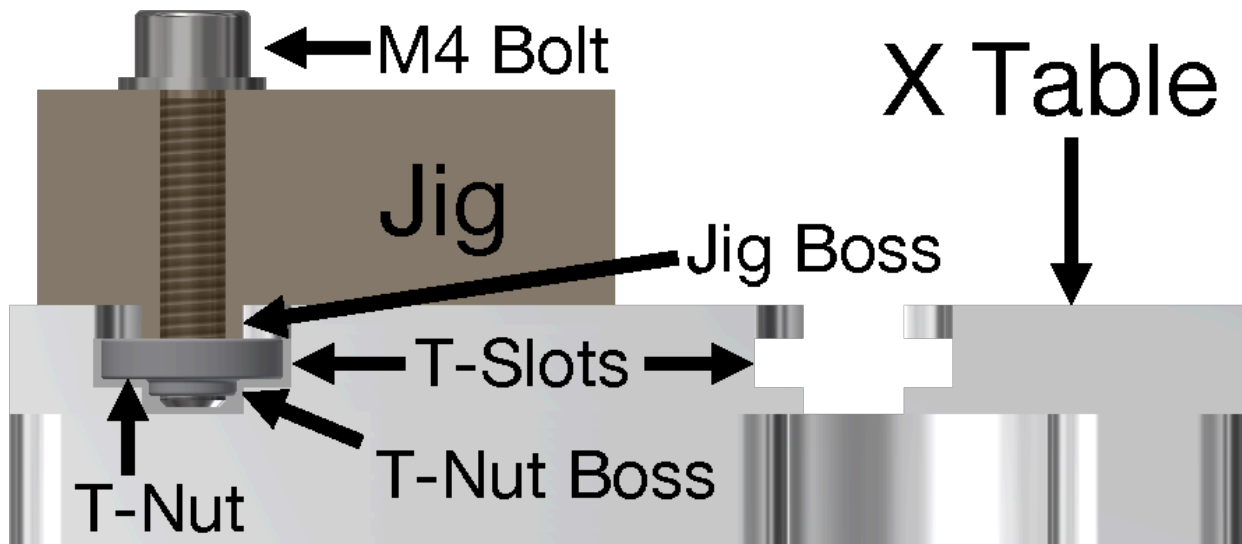
**Note:** Plastic fixtures can deform if the bolts are overtightened. Sufficient clamping force is achieved with slightly more than "finger tight" applied torque. Over-torqued plastic fixtures will fail prematurely.

**Note:** Before tightening the bolts, verify the cutting tool is not touching the work piece.

\*Most fixtures have a 6.1 mm (0.240") wide boss protruding 2mm from the bottom face (see graphic below). This boss aligns the fixture (and part) perpendicularly to the spindle. If the boss isn't recessed into the T-slot, then the M4 bolts likely won't reach the T-Nut. If the bolts don't reach the T-Nut, then verify the fixture bottom is flush with the X-Plate top.



The following profile view summarizes the above steps, showing a properly attached work piece:



X Table (profile view), showing Fixture Boss inserted into T-Slot Profile, and M4 bolt threaded through T-Nut.

## 6.2 - Removing T-Nuts

A: After machining is complete, disconnect the Probe Cable, unscrew the M4 bolts, then remove the work piece from the table.

**Caution:** The work piece may be hot after performing cutting operations. Verify the work piece is cool prior to grabbing.

B: Vacuum all chips from the X Table.

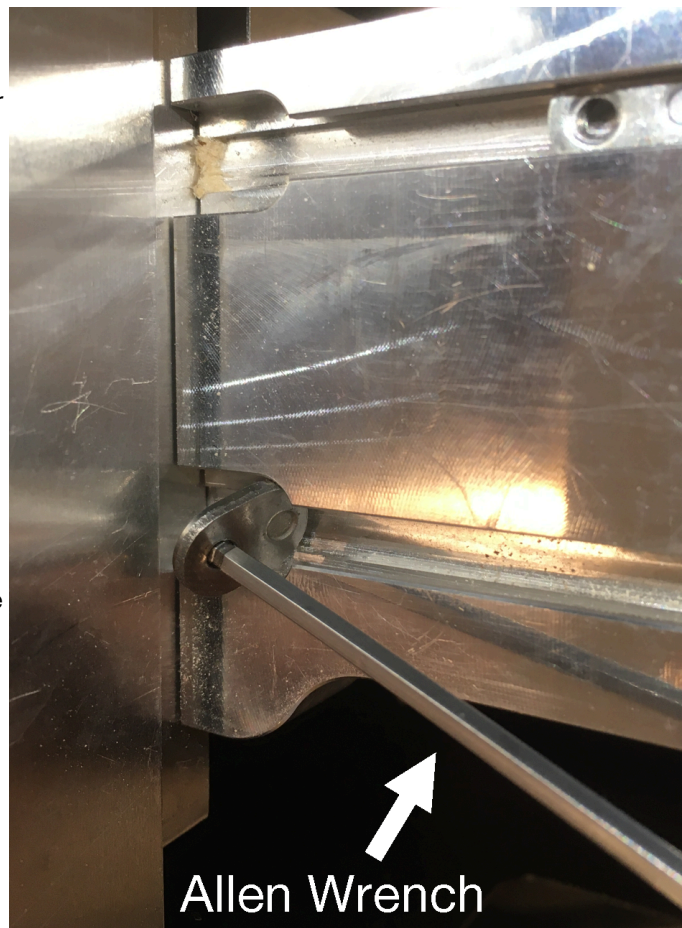
**Note:** Chips in the profile can prevent the T-Nuts from sliding.

**Note:** Unless a T-Nut is positioned all the way to the left (i.e. in the T-Slot Opening), the vacuum will not suck up the T-Nuts; they're secured by the profile.

C: Use the Allen wrench to slide each T-Nut left, until it reaches the T-Slot Opening.

D: Place the Allen wrench tip into the threaded portion of each T-Nut, then pivot the Allen wrench to rotate the T-Nut (right):

E: Push the (pivoted) T-Nut into the left aluminum chip guard, then lift up from the T-Slot Opening.





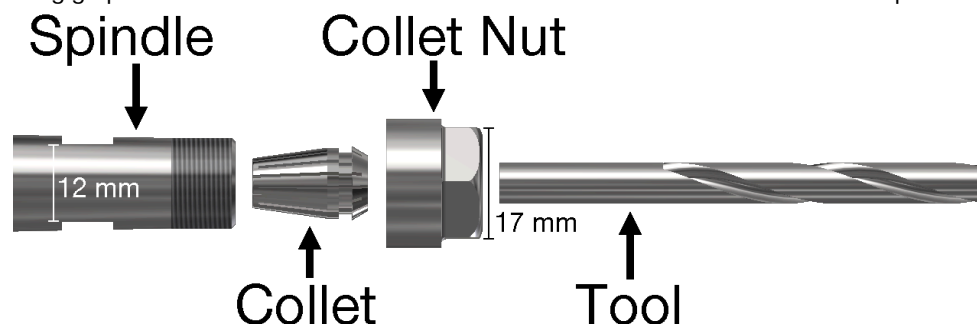
## 7 - Installing Tool

This section provides supplemental information and is a general overview on how to mechanically attach a cutting tool to Coast Runner's spindle. We've found that users unfamiliar with collet systems are more likely to succeed after reading this supplemental section. No action is required while reading this section; just learn the concept.

After reading this section, you should be able to install a tool, and later remove the tool and change collets. This section will seem trivial if you've used a collet system before. We still recommend reading it.

**Note:** These instructions assume a tool is already installed. If a tool isn't installed, skip to step J.

Use the following graphic and the instructions below to install a new tool into Coast Runner's spindle:



A: Plunge the spindle down to gain access to the 12 mm slot. There are several ways to do this:

- If you're running a prebuilt cut code in CRWrite, then the spindle is probably already in the correct position.
- If Coast Runner is unplugged, you can manually center the gantry and pull the spindle down.

**Note:** If you manually move the X Table, be sure to auto-level (\$L) before using Coast Runner.

- In CRWrite, open the Manual Operations window (i.e. click the joystick in the bottom right corner), then type the following commands into the Manual Entry text field:

\$H

G53 G21 G0 X0 Y-120

G53 Z-78

**Note:** Not required if you've previously homed this session.

**Note:** Moves X Table down and centers gantry.

**Note:** Plunges spindle to tool install location.

- You can also jog the spindle to an appropriate location in the Manual Operations window.

**Note:** Coast Runner is designed for use with tools up to 82.5 mm (3.25") long, that are fully seated into the spindle. Longer tools could crash into the X Table during the homing routine.

B: Place a 12 mm wrench onto the spindle's 12 mm slot.

**Warning:** Do not touch anything attached to the spindle unless the 12 mm wrench is securely held in place by hand. Severe cutting hazards exist if the 12 mm wrench is not present during cutting tool installation/removal.

**Note:** Coast Runner's electronics cannot overcome the stall torque applied by a firmly-held 12 mm wrench.

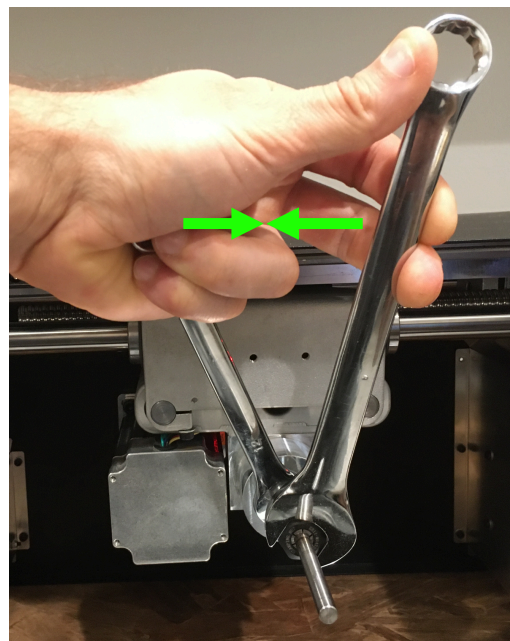
C: Rotate the 12 mm wrench to the 11-o'clock position.

D: Place a 17 mm wrench onto the collet nut (at 1-o'clock).

E: Squeeze both wrenches together to loosen the collet nut.

**Note:** Apply the force from one tool to the other, not directly into the spindle's mechanical frame.

**Note:** The nut will initially loosen, then tighten again.



Keep turning the collet nut counter-clockwise until it loosens a second time.

F: Remove the existing tool. If the tool won't move, gently wiggle the (sharp) tool tip until the collet releases.

**Warning:** Severe cutting hazards exist if the 12 mm wrench is not held in place during this step. The spindle could start spinning at any time if the wrench is not installed and held in place.

G: Unscrew the collet nut and verify the collet and collet nut are clean. Vacuum as needed.

**Note:** Metal chips reduce gripping force and increase runout.

**Note:** You can remove the 12 mm wrench after the cutting tool and collet nut are removed.

Remember to reinstall the 12 mm wrench prior to touching the spindle or installing a new tool.

H: If the replacement tool requires a different collet size, hold the collet nut stationary (e.g. on a table), then push the collet at an angle until it snaps out of the collet nut (see right ->):

**Note:** The collet won't pull straight out.

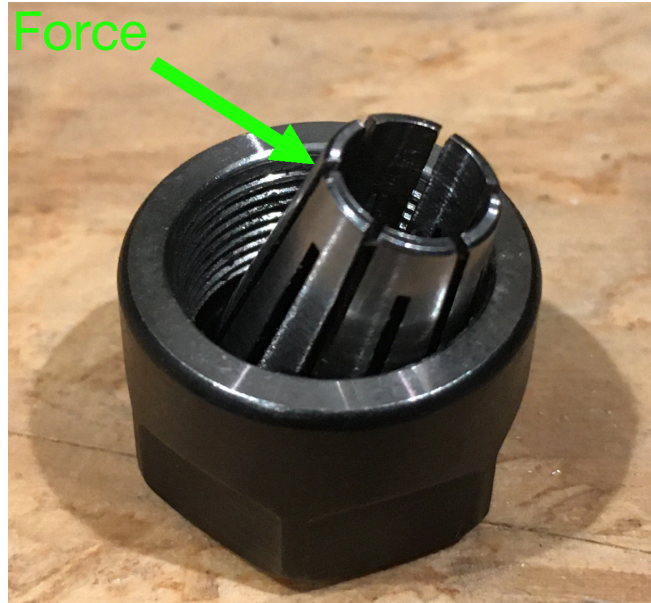
**Note:** Very little force is required.

**Note:** Still no luck? Rotate the nut 90°, then try again (the collet retaining ring is offset).

J: Install the correct collet by pressing it axially into the collet nut until it clicks into place.

**Note:** When properly installed, the collet is held in place by the collet nut. The collet is not fully installed if it falls from the collet nut when inverted... try again.

**Note:** Coast Runner accepts industry standard ER11 collets (sold separately).



K: Screw the collet nut a few turns onto the spindle, then insert the new tool into the collet.

**Warning:** Severe cutting hazards exist if the 12 mm wrench is not held in place during this step.

**Note:** The tool will not fit into the collet if the collet nut is too tight.

L: Push the tool fully into the spindle until it bottoms out.

**Note:** Tools that stick out too far could crash into the X Table while homing.

**Note:** Prior to probing, most prebuilt files position the Z axis assuming the tool is bottomed out.

M: Hold the 12 mm wrench stationary, then torque the 17 mm wrench clockwise to 30 foot-pounds.

**Note:** Insufficient torque can allow the tool to walk out while milling, ruining your work piece and tool.

**Note:** We used to say "it's difficult to over-tighten the collet nut," but over the years we've had a few meatheads strip the threads off entirely. In our testing, the (easily replaceable) collet permanently deforms around 75 foot pounds, whereas the spindle threads don't shear off until around 125 foot pounds. Given that wide range, we recommend erring on over-tightening the nut, within reason.

**Note:** 30 foot pounds is slightly more than "wrist tight." A 200 pound male shouldn't need to get the elbows or shoulders involved. If this isn't enough guidance, Horror Fraught's disposable torque wrench is \$19. Add a \$3 17 mm socket and teach yourself what 30 foot pounds feels like.

N: Remove the 17 mm wrench, then remove the 12 mm wrench.

## 8 - Installing Chip Cover

To install the clear polycarbonate chip cover, place the cover over the open bay such that the magnets securely hold the chip cover flat on all sides:

When properly inserted, the chip cover blocks the

Component	Location	Notes
Probe Connector	Inside build chamber, underneath enclosure	Barrel jack.
Probe LED	56 mm (~2.25") right from the Probe Connector	Green when lit. Difficult to see when not lit.
Probe Cable	Accessory box	Red cable with barrel connector and ring terminal.

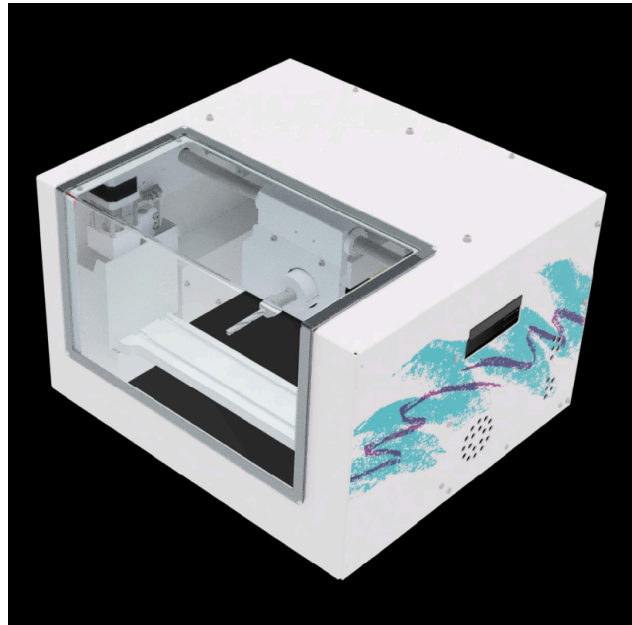
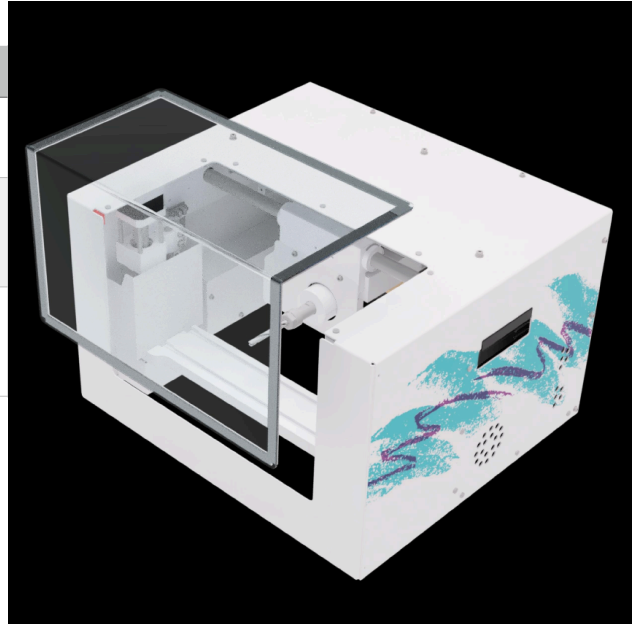
build opening, which prevents stray chips from leaving Coast Runner during milling operations. We recommend installing the chip cover prior to each machining operation.

**Caution:** The chip cover is not a substitute for safety glasses, and is not intended to contain an improperly secured workpiece or shattered end mill.

If desired, you can install the chip cover in the opposite orientation - with the long side on top and the short side covering the front - which leaves a 30 mm (~1.18") gap at the build opening bottom. This configuration allows a vacuum hose to remove chips while a machining operation is in progress.

**Warning:** Crush hazard. The X Table can crush, pinch and tear body parts. Do not place hands inside Coast Runner when the X Table is moving.

**Note:** Chips are more likely to escape Coast Runner in this configuration.



## 9 - Probing - Theory of Operation

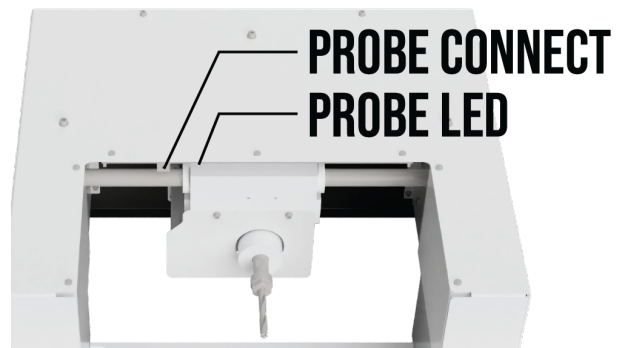
Probing is used to automatically determine where the work piece is located in 3D space, relative to the cutting tool.

The probe system consists of the following physical components:

**Note:** The entire probe circuit is safe to touch.

Looking inside the build chamber, the Probe Connector (left) and Probe LED (right) are circled:

*In this close-up, the Probe Cable is not plugged in, so the Probe LED is illuminated.*



To use Coast Runner's built-in probe, attach the Probe Cable's ring terminal to the work piece, then plug the Probe Cable's barrel jack into the Probe Connector. Specific attachment methods vary depending on the work piece.

Electrically, the Probe Cable's ring terminal is weakly pulled up to five volts. When the ring terminal isn't touching any metal inside Coast Runner, a voltage meter will measure ~4.7 VDC between the ring terminal and cutting tool.

However, when the Probe Cable touches any metal component inside Coast Runner (i.e. when the tool contacts the work piece), the signal is pulled down to zero volts, which tells Coast Runner that the probe is tripped. The Probe LED illuminates when the probe is tripped. The following table illustrates the various probe states:

Probe Cable Plugged into Probe Connector?	Probe Cable Shorted to Metal Inside Mill?	Probe Cable DC Voltage	Probe LED	Grbl Probe State	Grbl '?' Response
No	N/A	N/A	Illuminated	Tripped	P000
Yes	No	4.0 to 5.5	Off	NOT Tripped	0000
Yes	Yes	0.0 to 1.5	Illuminated	Tripped	P000

Let's verify the information in the above table is correct and the probe functions as designed:

A: Disconnect the Probe Cable from the Probe Connector. The LED turns on (the probe is tripped).

**Note:** Probe commands will fail if you forget to plug in the Probe Cable. This is a good thing.

B: Plug the red Probe Cable into the Probe Connector. Verify the Probe Cable's ring terminal isn't touching any metal component inside Coast Runner. The Probe LED turns off (the probe is not tripped).

C1: Touch the Probe Cable's ring terminal to the Spindle. The Probe LED turns on (the probe is tripped).

C2: Touch the Probe Cable's ring terminal to the X Table. The Probe LED turns on (the probe is tripped).

When the Probe Cable's ring terminal is connected to the work piece, it logically follows that the work piece must be electrically isolated from any metal component inside Coast Runner; otherwise the probe would always be tripped. Electrical isolation is typically achieved via plastic insulation between bed and fixture, or entirely plastic fixtures (other methods exist). Therefore, when the Probe Cable is connected to the work piece, the entire work piece floats up to around 4.7 volts (4.0 to 5.5 v).

Each time a probing cycle begins, Grbl first verifies that the Probe Cable voltage is above 4.0 volts. If the voltage is too low, Grbl returns an alarm and does not probe (see Appendix E for complete details).

**Note:** We recommend verifying the Probe LED is off prior to sending a probe command.

**Aside:** Grbl supports four probing modes (see Appendix A); we're describing the most common behavior.

Next, Grbl slowly moves the tool towards the part, while simultaneously monitoring the Probe Cable voltage. When the (grounded) cutting tool contacts the (5 volt) work piece, the Probe Cable voltage drops to zero volts (see above). After sensing this voltage drop, Grbl halts the probe motion and sends the probe trip point to the host.

**Note:** Stationary cutting tools do not have a uniform diameter. Therefore, for best probing accuracy the spindle must be rotating while probing.

**Note:** Make sure the metal surface to be probed - as well as the cutting tool - are free from stray chips and burrs. The probe will trip prematurely if the tool contacts a chip or burr prior to the intended surface.

Summarizing the above, Coast Runner's probe only works correctly when all of the following conditions are met:

- The Probe Cable must be plugged into the Probe Connector.

- The Probe Cable must be connected to the probed material.

- The probed material must not electrically contact any metal surface inside Coast Runner (e.g. X Table).

**Note:** Stray metal chips are conductive. We recommend judicious vacuuming between the work piece and X Table prior to each probing cycle.

- The probed material must be electrically conductive.

**Note:** Anodized aluminum surfaces are NOT electrically conductive. See "Probing Anodized Parts".

- The cutting tool must be electrically conductive.

**Note:** Some tool coatings are non-conductive.

The Probe LED turns on when at least one condition above is not met.

**Note:** The probe routine will immediately fail if the Probe LED is on prior to probing,

The Probe LED turns off when all of the above conditions are met.

**Note:** It's possible that the probe cycle will fail even though the Probe LED was initially off. For example, if the Probe Cable's ring terminal isn't connected to the work piece, then when the tool contacts the probed surface, the Probe Cable voltage won't drop to zero... and the tool will crash into the part.

**Note:** Optional sanity check: Before probing, verify the Probe LED is off, then temporarily touch a short wire scrap between the cutting tool and the probed surface. The probe is configured correctly if the Probe LED turns on when the scrap wire is connected (and turns off when removed).

We'll learn how to use g-code commands to probe later. For now just understand how the probe hardware works.



## 9.1 - Probing Anodized Parts

The following two facts illustrate why we do not recommend machining anodized parts:

- Coast Runner's probe system requires the cutting tool to electrically conduct to the probed surface.
- Anodized aluminum surfaces are NOT electrically conductive (but the raw aluminum underneath is).

Therefore, anodized aluminum work pieces will not trip Coast Runner's probe circuitry until the tool has cut through the anodized layer. While this layer is typically less than 0.003" thick, the probing routine is specifically designed to prevent the cutting tool from creating chips (since they introduce positional uncertainty to the probe results). Therefore, the typical probing routine is designed to:

Probe Routine Design Goal	Rationale	Why this is bad for cutting
Rotate tool at low RPM	Minimize chip quantity when contact occurs.	Increases torque required to remove each chip, which can stall the spindle.
Rotate tool the 'wrong' way	Prevent cutter engagement, and therefore chip creation.	The cutting flutes don't engage the material, but instead grind through the oxide layer.
Move probed axis slowly	Increase probe accuracy	It takes a while to grind through the oxide layer.

Given the above, we do not recommend probing anodized parts. However, many customers have successfully probed anodized parts in Coast Runner. We recommend the following tips:

- Use a file or sandpaper to remove the anodized surface around each probed face. Make sure to file/sand completely through the anodized layer... raw aluminum is gray/silver.

**Note:** When possible, we choose probing surfaces that are not visible.

- Verify that there are no loose chips on the probed surfaces, which will trip the probe prematurely.
- When the Probe Cable ring terminal is attached to the anodized part using a bolt, the shear forces between the threads are typically able to remove enough oxide for the thread-joint to become electrically conductive.
- Verify the probe will trip by temporarily touching a wire scrap between the tool and each sanded surface.  
**Note:** The probe will not trip if the Probe LED doesn't turn on when the scrap wire is attached.
- Modify the probing g-code routine to spin the tool the 'right' way (to cut through anodization).

## 9.2 - Probing Non-Conductive Parts

Coast Runner's probe system requires the cutting tool to electrically conduct to the probed surface. Therefore, you cannot directly probe non-conductive surfaces. However, there are a few ways to use the probe:

- Probe a conductive surface with a known offset to the part and/or fixture. Example: use a square aluminum block that is push-fit into a known location in the fixture. After probing the block, we mill the non-conductive part.
- Apply adhesive copper tape to the probed surfaces, then connect the Probe Cable to the copper tape.  
**Note:** Your code must account for the copper tape's thickness.
- Manually probe the part using the Manual Operation window and a piece of paper. If you're not familiar with this method, consult your favorite search engine (e.g. "machinist stock setup using paper shim").

# 10 - FAQ/Troubleshooting

## Q1: Where is Coast Runner's serial number?

Coast Runner doesn't have a serial number, but it does contain certain universal identifying marks that we may use to verify that we manufactured a particular machine.

## Q2: Why won't CRWrite connect to Coast Runner?

Is the E-stop button engaged? Rotate the E-stop clockwise to verify the E-stop knob is out (disengaged).  
Is another program connected to Coast Runner? If unsure, unplug and reconnect the USB cable.

If you've never successfully connected to Coast Runner, try a different computer, or try these OS-specific tasks:

### Windows:

Coast Runner's drivers are built into Windows 8 and later; no driver installation is required to use Coast Runner. However, Windows 7 doesn't natively support Coast Runner. During installation, CRWrite will automatically install the driver if granted administrator access. Otherwise, the driver must be manually installed (included on the USB flash drive).

### Mac OS:

Coast Runner's drivers are built-in to MacOS. In certain rare cases, the driver will not load correctly unless Coast runner is connected to the computer prior to booting; we've only seen this behavior while running 10.13 on a 5,1 Mac Pro's built-in USB ports. If CRWrite is unable to find the Coast Runner mill, try turning the computer off, plugging in Coast Runner, then turning the computer on.

## Q3: Why won't Coast Runner respond to motion commands?

- Use the Manual Control window to verify an alarm hasn't occurred (alarms halt all motion).
- Verify that the emergency stop switch isn't engaged.
- Coast Runner's onboard firmware prevents all motion until either homed (\$H) or unlocked (\$X).
- If the left side fan is not spinning, then Coast Runner isn't receiving power (the right fan is temperature controlled). Unplug the power and USB cables, wait 10 seconds, then reconnect. If the left fan is now spinning, then the power supply previously entered short circuit mode, but is working now.

## Q4: Why is Coast Runner moving to the wrong spot?

The default work coordinate system (WCS) is G54. If offsets are stored in G54, then Coast Runner will shift the origin by those values. To see if offsets are stored in G54, type `$#` and verify G54 is `[G54: 0.000,0.000,0.000]`. If not, send `$RST=#` to clear all offsets (see Appendix B: "\$RST" for more information).

**Note:** To avoid this confusion, we do not recommend storing offsets in WCS G54.

To force Coast Runner to move to an absolute machine position (based on the previous homing cycle), add `G53` to the beginning of your g-code line. G53 only applies to the line containing it (see Appendix A).

## Q5: Why does Coast Runner make noise when the USB cable is plugged in?

The X/Y/Z axes are positioned with stepper motors, which use micro-stepping to increase positioning resolution. Micro-stepping uses two high current PWM waveforms to place the motor between two discrete phases. PWM waveforms are variable frequency square waves. Square waves are mathematically infinite sinusoidal sums, which includes audible frequencies between 20 Hz & 20 kHz.

## Q6: Why are there two X stepper motors?

Coast Runner's horizontal spindle configuration doesn't allow a single, central linear X drive; the lead screw would punch through the build platform. Placing a single drive screw at one edge severely reduces the other edge's rigidity. Thus, two ball screws - and two steppers - are required to position the X Table (one on either side).

## Q7: Why does the spindle motor pitch change during warmup?

Bearings are conceptually simple, but are actually quite complicated. Coast Runner's spindle uses heavily preloaded, press-fit angular contact bearings to reduce runout. When the spindle is cold, non-uniform grease viscosity and metal constriction intermittently combine to require more power than Coast Runner's VFD controller is programmed to deliver. As the bearings heat, the metal expands and grease viscosity decreases, thus decreasing power lost as friction. Human ears are incredibly sensitive to pitch changes, but assuming the spindle is used within specification, the actual RPM value changes very little (less than 5%).

**Q8: How is spindle power measured?**

Most desktop CNC manufacturers list their machine's peak input power consumption, which occurs only when power is first applied to a stationary motor. This figure is typically MUCH higher than the motor's actual rated continuous cutting power consumption. For example, a certain '300 W' spindle used in other desktop CNC machines overheats in under 15 minutes when continuously loaded to 75 W using a dynamometer. Coast Runner's specified power is the continuous cutting energy transferable to the work piece, as measured by a Prony brake dynamometer.

**Q9: Why won't the spindle rotate?**

If everything else works - except for the spindle - then the most likely culprit is a latched over-current condition. This is a safety feature that occurs when something went wrong (e.g. the machine crashed, the part came loose, the cutting code toolpath was too aggressive, etc.). To clear a latched over-current condition, unplug BOTH the USB and power cables for ten seconds, then try again. If the problem persists, contact [support@coastrunner.net](mailto:support@coastrunner.net).

If you can hear the spindle motor, and yet the spindle doesn't spin, then replace the belt (see Appendix L).

**Q10: Why does Coast Runner keep starting and stopping?**

If the machine is running known-good g-code, verify the tool isn't worn out. As the tool wears, more energy is required to perform the same cutting task. If too much energy is pulled while cutting, then the dull tool can trip the over-current safety feature. With proper care, the consumable end mills and drills that ship with Coast Runner build kits should manufacture numerous parts, but require periodic replacement (available at [coastrunner.net](http://coastrunner.net)).

**Q11: Why did Coast Runner fail to home?**

Try homing each axis individually (\$HZ, \$HY, then \$HX).

**Note:** To prevent crashing the X Plate into the spindle, home Z prior to X.

If a particular axis fails to home, use a cotton swab to remove any chips/debris obstructing the optical path on that limit switch. A red LED on each limit switch illuminates when it is NOT tripped.

**Note:** Never adjust the limit switch tab positions; they're factory calibrated.

**Q12: Why is the X table binding/unlevel? - OR -****Q12: Why are the machined part features rotated (at an angle) from where they should be?**

Either the machine crashed, or the X Table was manually moved when the machine was unplugged. Regardless, the built-in leveling routine (\$L) should solve the problem (see "12 - Leveling X Table").

**Q13: Why did Coast Runner crash into the part while probing? - OR -****Q13: Why are the machined part features offset from where they should be?**

Verify the Probe Cable is plugged into the Probe Connector, and connected to the workpiece.

Verify the X Table and cutting tool are free from chips (see "Probing - Theory of Operation").

Verify the probed surface is conductive (see "9.1 Probing Anodized Parts" & "9.2 Probing Non-Conductive Parts").

**Q14: Are non-conductive fixtures required to mill parts?**

To use the built-in probe, the mounting method must prevent any probed metallic feature from conducting to the aluminum X Table. If the built-in probe isn't used (i.e. the part is manually aligned to the tool), then you can mount the part however you please (see "9 - Probing - Theory of Operation" & "19.3 - Designing Custom Fixtures").

**Q15: Why won't Coast Runner work?**

There's a ton going on inside Coast Runner. You might find the answer by reading this manual, which attempts to describe solutions to the most common issues we've observed over the past ten years.

Still stumped? Send our support team an email ([support@coastrunner.net](mailto:support@coastrunner.net)). Please include a detailed problem description... "It doesn't work" is incredibly difficult to troubleshoot. A linked video is often the best troubleshooting tool. We'll do everything we can to troubleshoot the issue remotely, but if we're not able to solve the problem we'll create a service RMA (see "RMA Policy" for complete details).



**Q16: Can I take Coast Runner apart?**

Sure! We invite the tinkerers among us to take a peak inside (see Appendix K). If something is already broken and you're already considering an RMA, attempting to repair the issue yourself won't increase the flat rate RMA cost (assuming you're unable to fix the problem). We'll even send you replacement parts, and assist you with the repair process. Note that some repairs are not possible except via RMA. The only caveats with this policy are:

- You must attempt the repair in good faith.
- You must send all the parts back (please place "extra" parts in a plastic bag taped inside the enclosure).
- To prevent shipping damage, all three axes should be sufficiently assembled to the internal aluminum frame.
- The internal frame should be attached to the outer enclosure (via QTY3 M4 bolts on the top cover).

**Q17: Why do some people recommend re-flashing the firmware prior to running each job?**

Re-flashing the firmware is generally unnecessary, as Grbl never modifies the firmware during operation. Our support team might suggest re-flashing the firmware to troubleshoot specific conditions, but otherwise we don't recommend it. Send us an email if you believe you have a valid use case that requires constant re-flashing. The only persistent values Grbl can modify are those stored in EEPROM (see next question). Technically these EEPROM values are not stored in firmware (i.e. flashing the hex file does not overwrite the EEPROM). However, after updating the firmware, if Grbl determines that an EEPROM value is invalid, then default values are restored.

**Note:** *On the contrary, no EEPROM reset occurs if all persistent EEPROM values are valid.*

Given the above, we recommend restoring the default EEPROM values from within Grbl, using the special `$RST=*` command. This command is much faster and almost certainly achieves the same outcome (versus re-flashing the firmware entirely, which *might* consequently restore the EEPROM).

Under normal use, Coast Runner should not require regular EEPROM resets. It's one thing to reset these values while developing new g-code. However, if you find that Coast Runner misbehaves after running existing cut code files, then we recommend having the developer contact us, so we can teach them how to achieve the same outcome without borking the EEPROM. The usual suspects are disabled soft limits and offsets stored in WCS G54.

**Note:** *We recommend reading Appendix D: "Inspecting Cutcode Files for Malicious Content".*

**Q18: Which values are persistent, even across power cycles?**

The following values are stored in EEPROM, and persist until a subsequent g-code line overwrites them:

- '\$\$' parameters **Note:** *Send `$RST=$` to restore defaults.*
- G54:G59 WCS offsets **Note:** *Send `$RST=#` to restore defaults.*
- G28/G30. **Note:** *Send `$RST=#` to restore defaults.*
- We also store certain factory calibration data in the EEPROM (neither `$RST` nor re-flashing touch these).

**Q19: What's the difference between a 'jig' and a 'fixture'?**

As it pertains to manufacturing:

- Fixtures hold parts stationary to allow cutting via moving tools; the cutting tool does not reference the fixture.
- Jigs hold parts, but must also somehow mechanically guide cutting tools (e.g. via a lever, alignment race, etc). The terms are used nearly synonymously in industry, with a prevalence towards 'jig', but Coast Runner attempts to use the correct term "fixture" in all circumstances, including this documentation.

**Q20: Why is Coast Runner a horizontal mill?**

Coast Runner's design was chosen to facilitate machining deep pockets in stock. The horizontal spindle configuration allows chips to more easily escape these pockets.

**Q21: Why are all specifications metric?**

The metric system is nearly universally adopted throughout the world because it's better in every conceivable way. Primarily, converting between units is logical (e.g. 1000 mm = 1 m = 0.001 km). Your failure to learn the metric system isn't a compelling reason not to use it. Coast Runner's default unit is mm.

**Q22: "But muh inches! This is America!"**

That's not a question, but OK, fine: Coast Runner supports the Imperial inch, too (see 'G20' in Appendix A).

**Q23: Why does the the Probe LED flash on and off while machining parts?**

The electronic probe circuit is always enabled (see "9 - Probing - Theory of Operation"). However, Grbl only monitors the probe circuit while executing a probe command. If the flashing LED bothers you, unplug the Probe Cable from the Probe Connector after the probing operation completes (the LED will now stay illuminated).

# 11 - Maintenance

Like most industrial machinery, Coast Runner requires periodic maintenance. With proper care, Coast Runner should remain operational for many years.

## 11.1 - Removing Chip Accumulation

The most important preventative maintenance is to remove waste chips with a vacuum after each milling operation. Waste chips occupy approximately fifteen times more volume than their uncut form. If allowed to accumulate, chips can work past guards and then contact the sensitive linear rails and ball screws, reducing bearing life. Chip accumulation can also affect Coast Runner's electronics, particularly when improperly cleaned with compressed air.

After milling, vacuum inside the machine via the front entry access. To simplify cleanup, Coast Runner lacks a lower panel. Once most chips are removed, tilt Coast Runner up slightly to gain additional vacuuming access. The goal is to vacuum away as many chips as possible to prevent contamination into sensitive components.

**Note:** Do not use compressed air to clean Coast Runner. Coast Runner's sensitive components are gravity-sealed from aluminum chips. Blowing air into Coast Runner will force chips past the sealed areas into sensitive components, decreasing Coast Runner's useful life.

**Note:** Inverting Coast Runner prior to cleaning could allow chips to bypass gravity seals and contact sensitive components. Thoroughly vacuum Coast Runner prior to inverting, shipping, or servicing.

We recommend periodically wiping chips from the shafts with a shop rag. The double sealed bearings tend to push chips towards the shaft ends. Chips that accumulate on the bearings and/or ballscrews can be gently brushed away with cotton swabs. Do not push directly into the chips, as this could force chips beyond the rubber seals. Instead, apply a gentle wiping motion around each bearing until all chips are captured in the cotton swab.

## 11.2 - Protection from Rust

Coast Runner is primarily metal. While most parts will not rust under even the most extreme conditions, a few steel parts are unprotected from oxidation and will develop cosmetic rust if stored in a humid environment. Parts with surface rust treatments will also oxidize if that treatment is compromised (e.g. due to chipping, scratches, etc).

Material	Inherent Oxidation Prevention Mechanism	Example Parts	Required Preventative Maintenance
Aluminum	Inherent, aluminum forms oxide with air	Internal structure, chip guards, motor bodies	None
Stainless steel	Chromium passivation	nuts/washers motor shafts	None
Powder coated steel	Coating isolates steel from oxygen	Outer enclosure	Apply touch-up paint to scratches, chips
Chrome plated steel	Chromium passivation	Shafts, ball screws	Remove accumulated debris with cloth/brush
Black oxide	Oil-impregnated coating	Bolts	None*
Chromoly steel	None	Spindle Spindle Body	Apply oil, particularly around collet threads

**Note:** To minimize oxidation formation, use/store Coast Runner in a climate controlled environment. If stored in a humid environment, consider placing a lit incandescent light bulb inside Coast Runner to slightly elevate component temperature (this will prevent water vapor from condensing on Coast Runner's metal surfaces).

**Note:** When stored and/or operated within specified humidity range, the black oxide bolts will not rust. Prolonged exposure to humidity above 80% will cause the bolts to rust, particularly in environments near salt water. Rusty bolts are a telltale sign that Coast Runner was stored and/or operated above specified humidity.

## 12 - Leveling X Table

Coast Runner can automatically level the X Table to within 0.002" across the entire top surface. In this context, 'level' means "make the X Table parallel to the gantry's YZ plane." To automatically level the table, you can either send the command '\$L', or in CRWrite you can press the "Auto-Level X Axis" button (Settings->Machine Actions). You can level Coast Runner whenever you want, although it is typically only necessary if the table position is manually adjusted while Coast Runner is unplugged, or if Coast Runner crashes during a milling operation.

**Note:** '\$L' automatically retracts the Z axis, so that the X Table doesn't crash into tools up to 3.25" long.

**Note:** The auto-level process only occurs when '\$L' is sent (e.g. \$H does not also call \$L).

**Note:** If the table is severely un-level, \$L can result in audible crashing noises. This is ok.

### 12.1 - \$L Theory of Operation

Coast Runner's X Table has two ballscrews (one on either end). During normal operation, both ballscrews are rotated synchronously, thus keeping the X Table level. However, if the X Table is manually repositioned (e.g. by hand, while the machine is unplugged), or if the the machine crashes/stalls, then the ballscrews might briefly spin asynchronously, which could un-level the X Table.

When the leveling routine is called, the X Table moves up until both limit switches on the X Table trip. The delta between these trip points is then compared to a pre-calibrated value. The absolute difference between these two values indicates how far from level the table is. To level the table, one ballscrew stepper motor is disabled, while the other is rotated, thus leveling the table. This measure-and-adjust process repeats three times.

### 12.2 - Manually Recalibrating 'Level'

Do not manually recalibrate 'level' unless at least one of the following is true:

- Running \$L does not accurately level the table (e.g. machined pockets aren't square, even after leveling).
- Coast Runner's factory calibration data was incorrectly overwritten (by using the '\$LS' command, see below)
- Coast Runner's X limit switches or tabs are replaced or physically repositioned.

**Note:** If the left limit switch or tab is repositioned, then it's possible Coast Runner will crash into the X end stop before the limit switch trips. If Coast Runner crashes after typing \$HX, then the X limit switch is almost certainly not tripping (move the tab closer to the limit switch, then try again).

Steps required to manually recalibrate 'level':

A: Launch CRWrite, then run "Manual\_X\_Calibration.crproj".

**Note:** This file is available by contacting support@coastrunner.net

B: Select "Level Table - Dial Indicator". Follow the steps presented to manually level the table.

**Note:** The dial indicator must be mounted to a 3D printable jig. If you do not have a 3D printer, or if you do not have a dial indicator, then choose "Level Table - Feeler Gauge" instead (less accurate).

**Caution:** Do not home Coast Runner with the dial indicator installed; it will crash into the right chip guard.

C: After finishing the steps listed in the .crproj file, click on the Manual Operation window (joystick bottom right).

D: Type '\$LS' into the manual text entry box. Coast Runner will retract Z and then store the X offset.

**Note:** See "12.3 - \$LS Theory of Operation" for more information.

E: Quit and Relaunch CRWrite, then run "Manual\_X\_Calibration.crproj" (same file as before).

F: Select "Verify Level - Dial Indicator". Follow the steps presented to verify \$L properly levels the table.

**Note:** If you do not have a dial indicator, then choose "Verify Level - Feeler Gauge" instead.

### 12.3 - \$LS Theory of Operation

'\$LS' is a power user feature. Do not use \$LS until you fully understand its behavior. Entering '\$LS' causes Coast Runner to move the X Table towards X home (up) until both limit switches have tripped. The delta between the two trip points is then permanently stored in memory (until the next time \$LS is called). Put another way, the '\$LS' command tells Coast Runner "the table is level now; store the delta between both X limit switch trip points."

Future '\$L' calls (to level the table) use the calibration data stored during the last \$LS call. If the last \$LS call occurred while the table was not level, then all future \$L calls (to 'level' the table) will un-level the table instead. Therefore, \$L is only as accurate as the last \$LS command sent. Therefore, you should only send '\$LS' immediately after manually leveling the table (ideally with a dial indicator, but a feeler gauge will suffice).

**Note:** Sending '\$LS' overwrites the factory X Table limit offset calibration data.

**Note:** Sending '\$LS' is not allowed within prebuilt files. This is a security feature to prevent malfeasants from intentionally overwriting the factory calibration data (i.e. by hiding \$LS within a prebuilt file).

## 13 - Updating Firmware

### 13.1 - Automatic Updates

Firmware updates are available using CRWrite's built-in update tool. With user consent, the firmware update process is otherwise automatic. Therefore, the manual firmware update method is unnecessary under normal conditions.

### 13.2 - Manual Updates

Use the manual firmware update process to:

- Recover after a failed firmware update attempt (e.g. if the power goes out during an update).
- Troubleshoot if CRWrite is unable to connect to Coast Runner.
- Restore a previous firmware version (the automated process typically only installs the latest supported version).
- Run your own custom firmware (way beyond the scope of this manual).

Coast Runner's primary PCB is an open source Arduino Uno clone. Therefore, the simplest method to manually update the firmware is to use existing tools for that platform. We recommend the following process:

A: Verify Coast Runner's USB cable is plugged into the host.

B: Verify the E-Stop is not engaged. The firmware will not update if the E-Stop is engaged.

C: Close all serial connections to Coast Runner (e.g. quit CRWrite, close any 3rd party programs, etc).

D: Find a CR\_firmware.hex file (available at [coastrunner.net](http://coastrunner.net))

**Note:** To troubleshoot spindle issues, choose an older firmware version (explanation below).

D: Open an Arduino compatible hex uploader. We recommend:

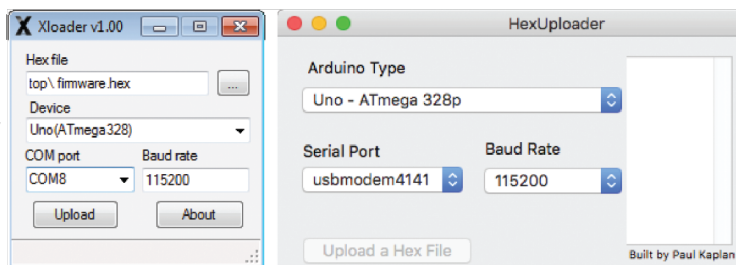
(Windows): XLoader

(Mac OS): HexUploader

E: Configure the hex uploader as shown:

**Note:** The COM/Serail Port number syntax is based on your particular system configuration.

F: Click Upload.



G: Send `$RST=*` to reset Grbl's persistent parameters (see Appendix A).

**Note:** The way parameters are stored in EEPROM might change between firmware versions. Resetting the parameters ensures the EEPROM values are read from the correct location.

The Manual Update process only replaces the firmware on the primary 328p processor. A secondary 32M1 processor (that controls the spindle) is not updated. Therefore, after performing a Manual Update, we recommend performing an Automatic Update in CRWrite, which will update the firmware on the secondary processor.

**Note:** The spindle might not work - or might work incorrectly - if the Automatic Update is not performed.

**Note:** Manual spindle firmware updates are possible, but beyond the scope of this manual

### 13.3 - Writing Custom Firmware

This is way beyond the scope of this manual. We may charge you a consulting fee to support issues pertaining to custom firmware development/troubleshooting.

## 14 - CRWrite - Privacy

Except as noted in this section, CRWrite does not transmit information via the internet.

**Note:** CRWrite does not require an internet connection to run.

### 14.1 - CRWrite Log File

During normal operation, CRWrite stores every command sent to Coast Runner in a log file, which is located:

Operating System	CRWrite Log File Path
Mac OS	~/CRWrite/logs
Windows	%userprofile%\CRWrite\logs

CRWrite will never send this log file over the internet, unless you explicitly choose to include it while submitting a service request from within CRWrite (i.e. by clicking Support>Contact Us, and then checking the box "Include logs and data that might help us solve your problem").

We may ask you to manually send the log file to us for debugging purposes. If we request the log file, please compress the file (e.g. .zip, .7z, etc.) to reduce the file size. If desired, you can send only the log file portions that you feel are relevant to the troubleshooting issue at hand. Typically this would include at least the last several dozen lines in the log file. We do not retain these log files after troubleshooting a problem.

If desired, this log file can be deleted at any time. However, CRWrite will automatically recreate a new, empty log file the next time any command is sent to Coast Runner.

**Note:** Every entry CRWrite places into the log file persists until the line - or the complete file - is deleted.

### 14.2 - Other Information CRWrite Stores Locally

CRWrite may also store certain identifying information in the following locations:

Operating System	File Path
Mac OS	~/Library/Application Support/CRwrite <i>n</i> (where <i>n</i> is the CRWrite version) ~/Library/Application Support/Caches/crwrite-updater ~/Library/Application Support/CrashReporter/CRWrite____.plist ~/CRWrite/
Windows	%userprofile%\AppData\Roaming\CRWrite <i>n</i> (where <i>n</i> is the CRWrite version) %userprofile%\AppData\Local\crwrite-updater %userprofile%\CRWrite

If desired, you may delete these directories as long as CRWrite is not presently running.

**Note:** CRWrite may automatically recreate these file paths when running.

### 14.3 - Other Information CRWrite sends over the Internet

CRWrite automatically connects to the internet to check for software and firmware updates. You can choose whether or not to install any updates. During this process, CRWrite sends the following information to our servers:

- The presently installed CRWrite version, so that we can see if an update is available.
- The operating system version (e.g. Win7x64SP1), so that we can send the correct update file.
- The CRWrite version string, returned by typing '\$I' (e.g. [grbl:1.1h CR:3C PCB:3B YMD:20200312]).

When accessing the built-in Support Portal, CRWrite may access the internet to retrieve information required to satisfy your support request.

## 15 - RMA Policy

This RMA Policy applies to all customer requests for Return Merchandise Authorization (RMA), and is in addition to those found in the Coast Runner Terms and Conditions of Sale (Terms of Sale). Where there is a discrepancy between the two, the Terms of Sale take precedence and supersede this RMA Policy. We may modify this RMA Policy at our sole discretion at any time. The most recent terms are available at: [coastrunner.net/terms/](http://coastrunner.net/terms/)

We will not accept any product for service or repair without prior authorization. To obtain authorization:

- A: Contact our support team: [support@coastrunner.net](mailto:support@coastrunner.net)
- B: If we agree that a return is required, we will send you an RMA Application Form and an RMA Number.
- C: Complete the RMA Application Form and include it with the returned product.
- D: To prevent shipping damage, Coast Runner units must ship in their original packaging. If the original packaging is no longer available, please request a new box PRIOR TO SENDING AN RMA.
- E: Write the RMA Number on the shipping label or packaging.

**Note:** We must receive the product within 30 days after issuing an RMA Number.

WE ARE NOT OBLIGATED TO ACCEPT ANY RETURNED PRODUCT not in compliance with this RMA Policy. NON-COMPLIANT RETURNED PRODUCTS MAY BE rejected and returned to CUSTOMER freight collect.

Each non-functional/defective product requires a separate RMA. Only send the nonfunctional product described in the RMA Application Form. Do not include functional peripherals unless requested (e.g. cutting tools, fixtures, probe cable, etc); they may or may not be returned.

Coast Runner products are sold without warranty. A flat rate Service Charge covers all parts and labor required to replace materials deemed defective through normal use. At our discretion - and on a case-by-case basis - we may offer to waive some or all incurred Service Charges. RMA repair work comes without warranty.

For damage that we deem occurred beyond normal use, our RMA Department will notify Customer prior to charging more than the Service Charge, and will provide Customer with an estimate regarding the cost of such service. We reserve the right to charge the Customer for parts, labor, and all other expenses, if we determine that:

- Consumable components returned with product require replacement, or;
- Required repair work is due to: misuse, abuse, alteration, improper installation, incorrect repair resulting in damage, negligent use, improper handling, or inadequate protection during transportation, or;
- Customer intentionally misrepresented information on the RMA Application Form to make it appear that the product required RMA due to normal use, or;
- No defect is found.

If no defect is found, our RMA Department will attempt to contact Customer to obtain additional information to reproduce the defect. If we are unable to obtain further diagnostic information from the Customer within five business days, we will assume the product is operating correctly and will return it without additional testing.

Unless otherwise noted, expected turn-around time to service or repair products is thirty (30) business days from the day we receive the RMA. Our failure to repair or replace the product within the turn-around time does not breach this RMA Policy.

Coast Runner shall not be liable for any delay in performance directly or indirectly caused by or resulting from acts of nature, fire, flood, accident, riot, war, government intervention, quarantine, embargo, strike, equipment failure, late deliveries by suppliers, or other difficulties which are beyond our immediate control.

**USERS RUNNING PREBUILT FILES CAN STOP READING THE MANUAL HERE.**

## 16 - Technical Overview

Coast Runner is an open source CNC machine designed to mill plastic, aluminum and steel objects. Whereas 3D printers additively manufacture objects by depositing material in layers, Coast Runner subtractively manufactures objects by cutting material away with a tool. Subtractive manufacturing is somewhat more difficult than additive manufacturing because the machine must accurately determine where an existing part resides in 3D space, and then must remain rigid enough to physically cut away material with specific cutting tools using a specific toolpath.

The following table highlights difficulties inherent to subtractive manufacturing:

Difficulty	Additive Manufacturing (3D Printing)	Subtractive Manufacturing (Coast Runner)
Part Mounting	Not required	Fixtures (universal clamps, vise, custom moulds, etc)
Initial Geometry	Not required	Required to prevent crashing
Part Probing	Not required	Cutting tool used as probe
Machine Rigidity	Not required	Aluminum unibody frame
Cutting Tools	Not required	ER-11 collet system Step-by-step setup
Tool Path Creation	Simple, automated	Designed or shared
File Distribution	Universal '.stl' format	Less universal distribution package

Why bother with difficult subtractive CNC manufacturing? 3D printing is great for prototyping, but CNC manufacture creates objects equivalent to those traditionally available only through regulated commerce. Coast Runner fulfills the promise 3D printing dreams of, today: easily manufacturing durable products in the privacy of your own home.

At its core, Coast Runner is a general purpose horizontal CNC machine, designed to manufacture parts from existing raw stock, using industry-standard g-code commands. By placing these commands into distributable .cproj files, Coast Runner becomes a means of production for the casual hobbyist. We have developed several .cproj files, but generally leave the machine's wider manufacturing capabilities to its vibrant artisan community. We strongly encourage developers to distribute their creations with the community.

---

The remaining sections in this manual teach power users how Coast Runner operates. You can then use that information to create custom milling files. There's a lot to learn, so buckle up! We recommend content creators read the entire manual prior to attempting to write and run their own gcode. Along the way we'll highlight numerous common pitfalls our users have reported over the past years.

The general layout is as follows:

- First we'll learn how to connect to Coast Runner.
- Next we'll learn how to interact with Grbl (the firmware running on Coast Runner).
- Then we'll learn several g-code concepts.

# 17 - Connecting to Coast Runner

There are several ways to connect to Coast Runner:

Interface	Ease-of-Use	Description	Recommended Use Case
CRWrite	Simple	Run a properly-formatted .crproj file (see "5 - QuickStart").	Validating finished code
	Intermediate	Use the Manual Operations window (similar to a standard CNC MDI entry).	Single-line entries Run single g-code files.
3rd Party Grbl Controller	Intermediate	Several 3rd-party Grbl-specific controller programs exist. Examples include grblController, Candle, Universal G-Code Sender, Carbide Motion, bCNC, Grbl-Panel, GCode Teleporter, Easel, and many others.	Multi-Line G-Code (i.e. cutting code generated by CAM program)
Any Serial Terminal	Advanced	Power users can connect to Coast Runner with any serial terminal. This manual shows how to connect to mill using the Arduino Serial Window, but any program that can send and receive serial commands will work.	Power Users

Some interfaces hold your hand more than others. Choose whichever interface you prefer.

**Note:** Only one interface can be used at a time. Close any open session before opening another.

In the pages below we'll explore the merits of each interface in more detail. Keep in mind that several concepts mentioned below haven't been formally explained yet; don't worry about full comprehension right now.

## 17.1 - Connecting to Coast Runner - Run .crproj File

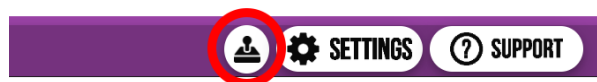
To use this method, create a .crproj file (as described later) and run it in CRWrite. This is identical to the method outlined in "Software - QuickStart". Initial code development typically doesn't occur using this method, as CRWrite's hand-holding process limits your ability to change g-code on the fly. This method is primarily used to test properly packaged, nearly-finished code, prior to distribution.

## 17.2 - Connecting to Coast Runner - CRWrite Manual Operations window

When developing new milling code, we recommend using the Manual Operations window to determine the correct motion commands to move each axis to successfully probe your part. As each g-code command is determined, that command is copied into a separate text file, which is then saved into the working directory. At any time during development, you can select and execute these files within the Manual Operations window.

**Note:** The Manual Operations window does not create g-code for the actual cutting geometry.

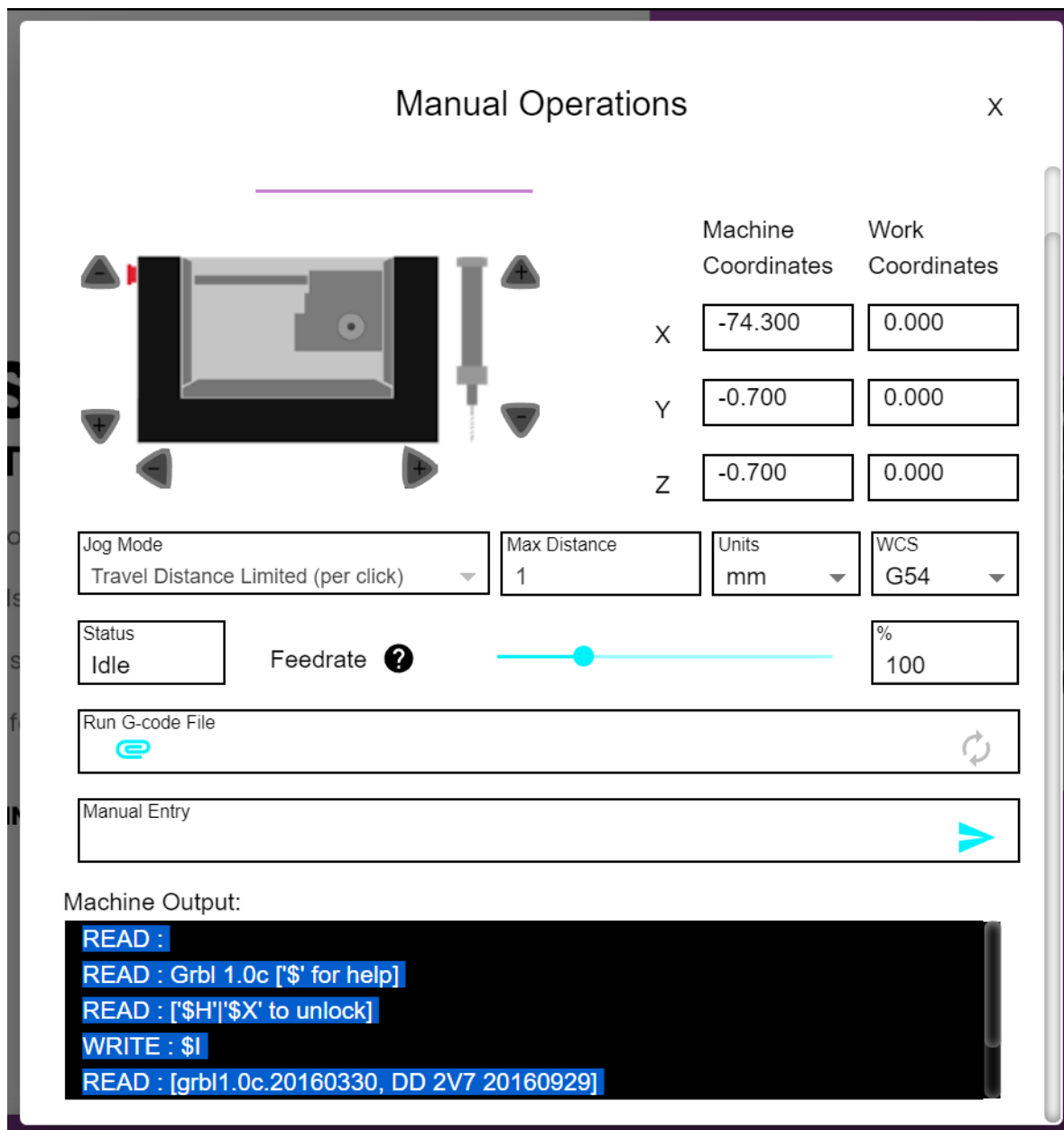
To access CRWrite's Manual Operations window, launch CRWrite, wait for CRWrite to connect to Coast Runner, then click the 'joystick' icon in the bottom right corner:





The following window appears after clicking the joystick in CRWrite:

**Note:** A popup will appear if Coast Runner hasn't been homed yet this session. Select "Home Now".

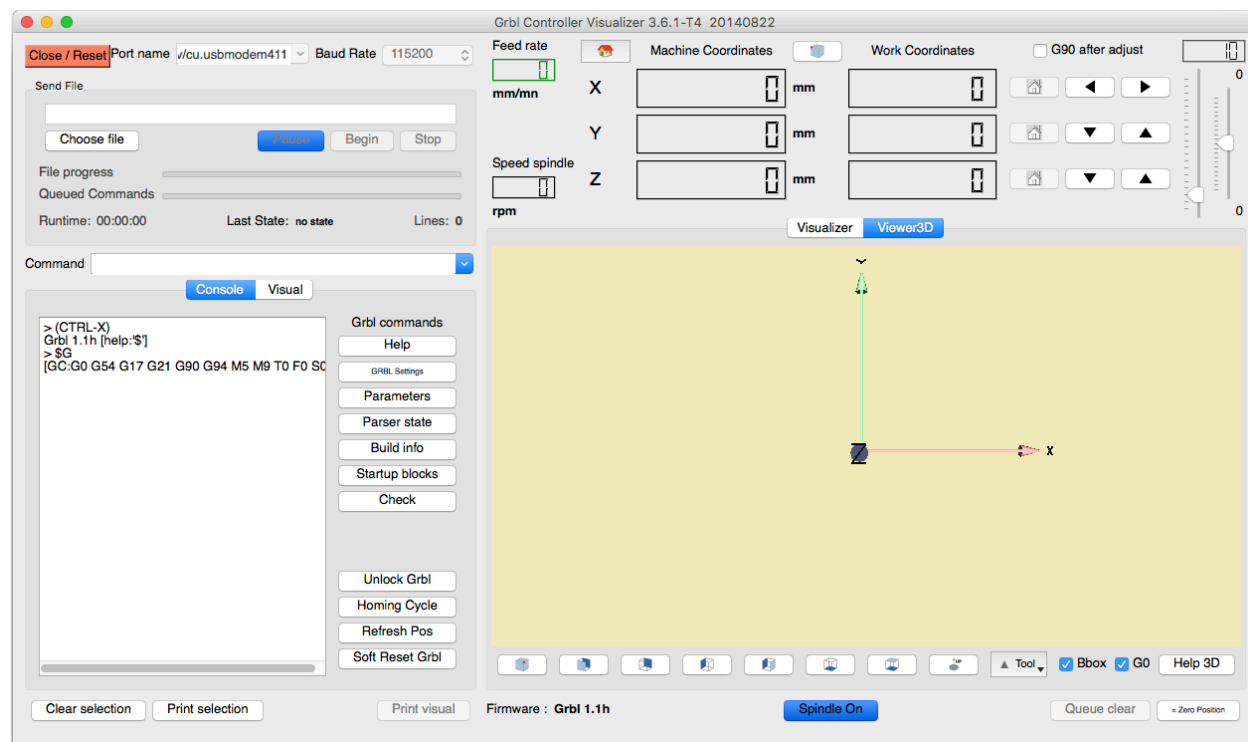


The Manual Operations window allows complete Coast Runner control, with common tasks only a click away:

- X/Y/Z linear travel, via clicking the arrow head graphic by each axis, or via configurable key bindings.
- Configuring Jogging Mode behavior (move continuously, or move up to specified distance).
- Selecting which WCS to display. Changing this value updates Grbl's modal WCS state as well.
- Configuring the units (mm or inch) used to display machine/work coordinates, and max jogging distance.
- For everything else, any valid g-code line can be manually typed into the "Manual Entry" text box.

### 17.3 - Connecting to Coast Runner - 3rd Party Grbl-Compatible Controller

Coast Runner is compatible with many existing 3rd-party Grbl-compatible programs. We recommend Zapmaker's grblController, even though it hasn't been updated since 2014. The latest version is included on the USB drive that ships with Coast Runner.



*Zapmaker's grblController, connected to a Coast Runner unit.*

While Coast Runner is generally compatible with any Grbl-compliant controller, this manual doesn't describe how to use them in any particular detail; refer to each specific program's documentation. Many of these programs obfuscate Grbl's serial responses, which can impede troubleshooting. For example, sending ? in grblController doesn't return any result, which is super annoying when troubleshooting machine behavior.

---

#### 17.4.1 - Connecting to Coast Runner - Serial Terminal (Arduino)

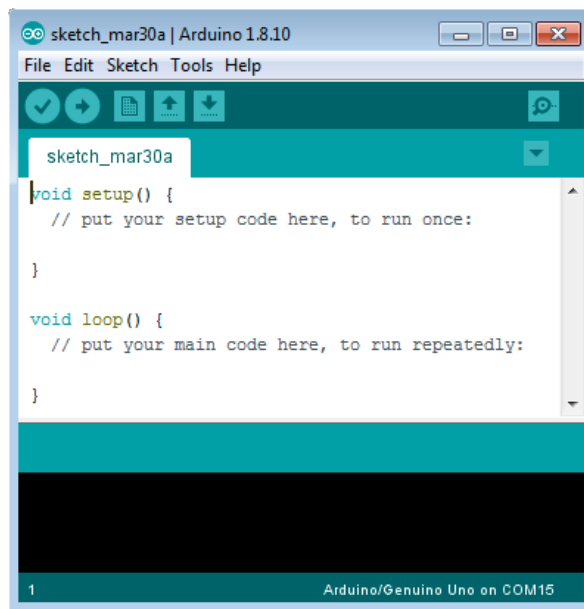
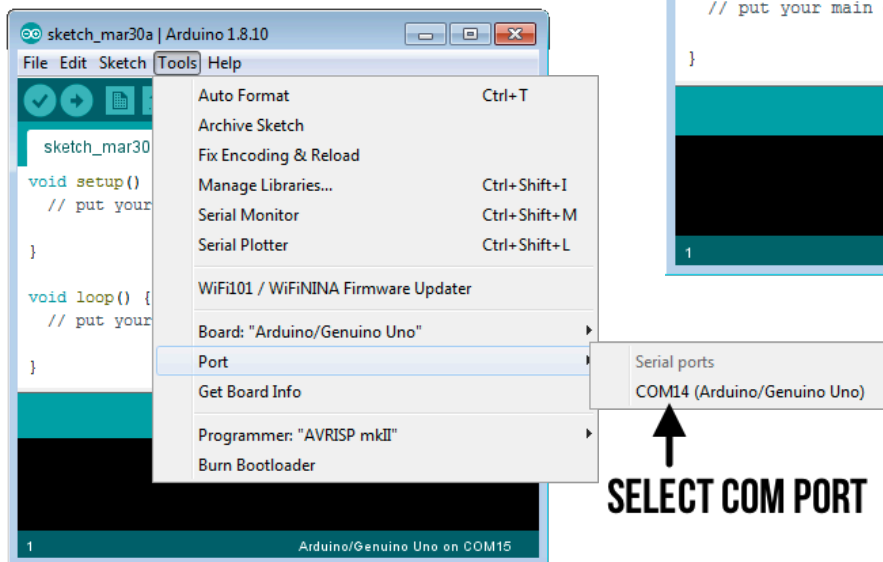
Power Users wanting absolute control can use any serial terminal interface to communicate with Coast Runner. This is typically only necessary if you're troubleshooting the actual serial protocol itself, or if the other interfacing methods are hiding Grbl's serial responses. We don't recommend using a Serial Terminal unless you really know what you're doing. There's absolutely zero abstraction, so you'll need to completely understand how Grbl works.

Due to its cross-compatibility and general ease-of-use, we've chosen to explain how to use the Arduino Serial Window. Download and install the latest Arduino software here:

<https://www.arduino.cc/en/Main/Software>

Once installed, launch Arduino. You'll be greeted with a window similar to the one shown at right. We won't actually use this window except to configure the serial port, and then to open the actual serial window that connects to Grbl.

To connect to Coast Runner's serial COM port, click Tools > Port > COMxx:



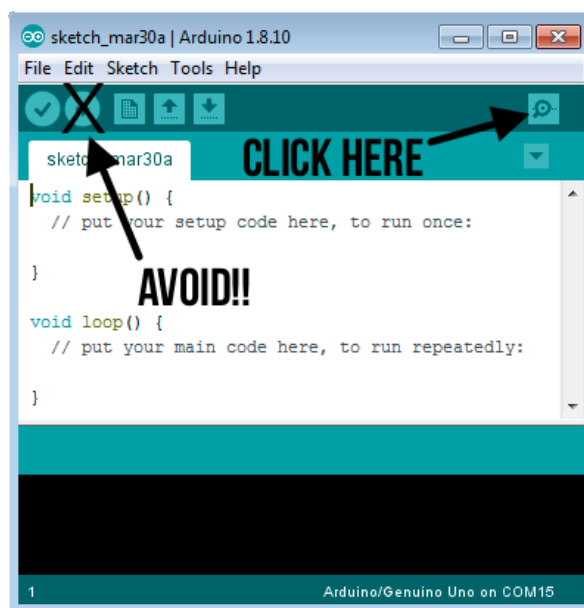
Click the magnifying glass icon (in the top right corner).

**Note:** Do NOT click the 'Play' button (marked "AVOID!!"). Clicking 'Play' will overwrite the Grbl firmware with an empty program, which will prevent CRWrite from recognizing Coast Runner. You'll need to reinstall the firmware using a 3rd party firmware flashing utility (see "13.2 - Manual Updates").

Clicking the magnifying glass icon opens a new window (shown next page). This is Arduino's low-level Serial Monitor Window, which shows all data sent between Coast Runner and the host computer. We call this the **Serial Window** (shown next page).

**Note:** Only one program can connect to Coast Runner at a time (e.g. CRWrite must be closed).

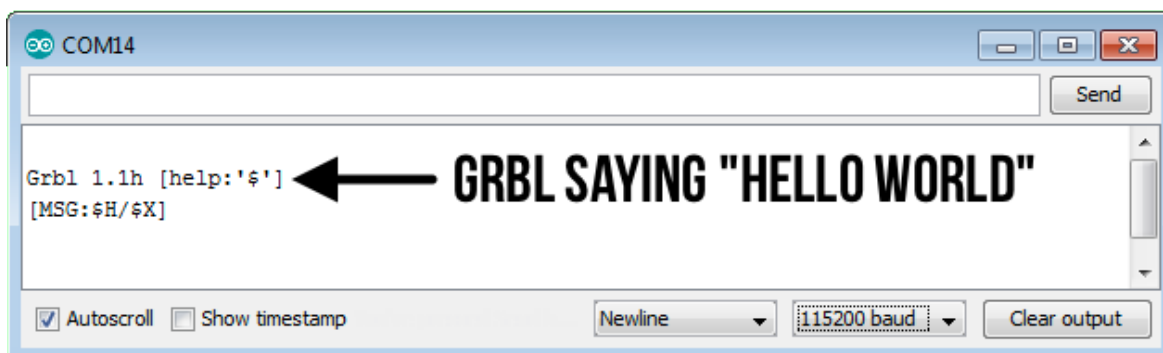
**Note:** Verify the hardware E-stop is disengaged.



As the Serial Window opens, the computer attempts to establish a low level serial connection to Grbl (the underlying firmware running on Coast Runner). If the Serial Window successfully connects to Grbl, then Grbl will attempt to display a text message in the lower text box (e.g. "□□").



That gibberish isn't the "hello world" text we're expecting. Grbl is talking at a different speed than the Serial Window is expecting. To fix this, change the default baud rate by clicking on the drop-down menu that says "9600 baud" and then selecting "115200 baud" (see above image). Behind the scenes, Arduino will re-establish the serial connection at the specified baud rate. Grbl should now greet you with the following message:



---

#### 17.4.2 - Connecting to Coast Runner - Serial Terminal (PTY)

This is hardcore mode. We're not going to hold your hand much here. You're probably fine with that. First, list devices: `ls /dev/tty.*`, which should return Coast Runner's handle (e.g. `/dev/tty.usbmodem411`). Next, connect to Grbl with PTY: `screen /dev/tty.usbmodem411 115200`. Congrats, you're connected! To close the connection, hold ctrl+a, then type :quit.

**Note:** Full interface implementation details are beyond the scope of this manual (see <https://github.com/gnea/Grbl/wiki/Grbl-v1.1-Interface>).

---

#### 17.5 - Connecting to Coast Runner - Conclusion

We've now explored the various methods to connect to Grbl. Again, don't worry if things are unclear. At this point, the only thing that should be clear is that there's a huge chasm between casually running preconfigured .cproj files, versus actually designing them. Read on and (hopefully) things will become more clear.

For the time being, we suggest connecting to Coast Runner using CRWrite's built in "Manual Operations" window.

## 18 - Interacting with Grbl - Hands on Example

Coast Runner uses the (excellent) open source 'Grbl' motion controller ([github.com/gnea/grbl](https://github.com/gnea/grbl)), which has become the de-facto standard for desktop CNC manufacturing. Grbl supports many industry-standard g-code commands, which we'll explore in the pages below (see Appendix A for a complete list).

**Note:** Specifically, Coast Runner is running a fork of Grbl which is a nearly identical Grbl fork, except as noted in Appendix H. For now, understanding the slight differences between Grbl and our fork is unnecessary, but we'd be remiss to not mention that minor differences do exist. Power users may wish to diff the above-mentioned directories for an exhaustive list of changes.

### 18.1 - So What is Grbl, Exactly?

Grbl is a g-code processor that resides in firmware on Coast Runner's digital PCB. Specifically, Grbl runs on one of the three microprocessors contained within Coast Runner. To be clear, Grbl isn't running on your computer.

The following statements are conceptually valid:

"I opened a serial connection to Grbl."

"Grbl returned an error when I sent G0G1."

"I manually upgraded the Grbl firmware running on Coast Runner."

From a machinist's perspective, Grbl's sole function is to interpret your expertly crafted g-code lines. Looking under the hood, whenever we send a serial command to Coast Runner, Grbl validates the command, processes the instruction(s), performs the requested action(s) (if any), and then sends a response back to the host. Behind the scenes Grbl is doing a whole lot more, but those details are unimportant for understanding this section.

Due to Grbl's extremely limited resources (e.g. 2 KB total RAM), only a dozen or so commands can be sent to Grbl at a time. Therefore, we rely on an external host computer to "feed" Grbl new lines of G-code as old lines are consumed. In practice these limitations shouldn't matter to you, as any program that can automatically feed multiple lines to Grbl is going to have a built-in flow control algorithm. Some of these algorithms are better than others; CRWrite's is particularly optimized, and thus is less prone to stuttering when sent numerous small motion commands over a short time period. For most developers, that's as much as you need to know about what Grbl is.

### 18.2 - Interacting with Grbl - Getting Started

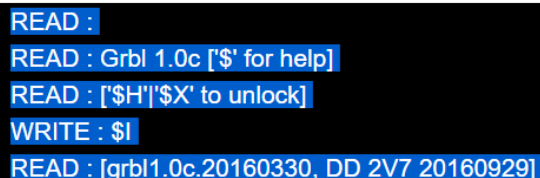
Grbl can tell us quite a bit about itself, including it's current status, hardware & firmware versions, stored settings, etc. For now, don't worry about understanding every gritty detail in this section; we'll cover more details later.

Each time we connect to Grbl - using any method outlined in Section 17 - we should see the welcome text:

```
Grbl 1.1h [help:'$'].
```

If you don't see that text, then something is wrong (e.g. baud rate, E-stop engaged, Coast Runner unplugged, etc). When using CRWrite's Manual operation window, we can see the above default response on the second line:

Machine Output:



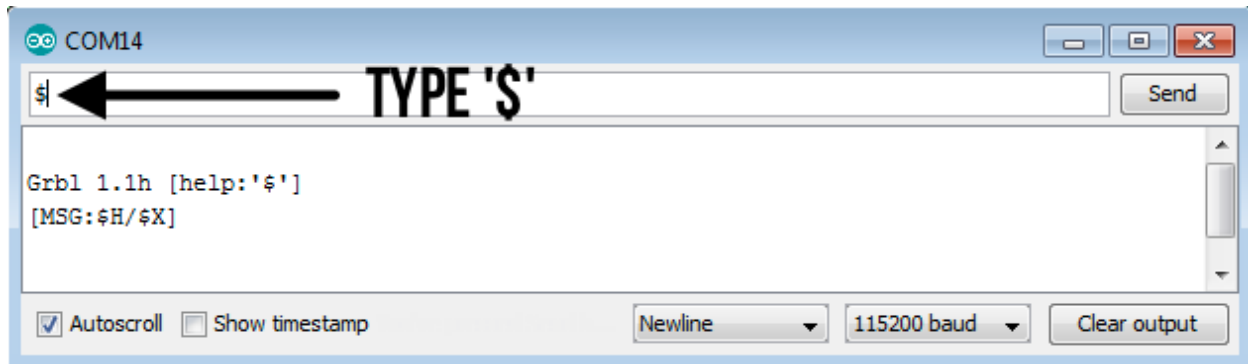
```
READ :  
READ : Grbl 1.0c ['$' for help]  
READ : ['$H','$X' to unlock]  
WRITE : $I  
READ : [grbl1.0c.20160330, DD 2V7 20160929]
```

We can see that after reading Grbl's welcome message, CRWrite automatically sends `$I`, which asks Grbl to identify itself. CRWrite performs this action so it can determine which Coast Runner version it's talking to.

**Note:** Remember, you can connect to Grbl using any of the interfaces mentioned previously. The graphic above shows a connection using CRWrite, whereas the graphic below shows Arduino's Serial Monitor.

Alright, now that we're connected, it's time to buckle up... we're about to dive headfirst into Grbl!

The welcome text suggests that typing '\$' will display a help menu:

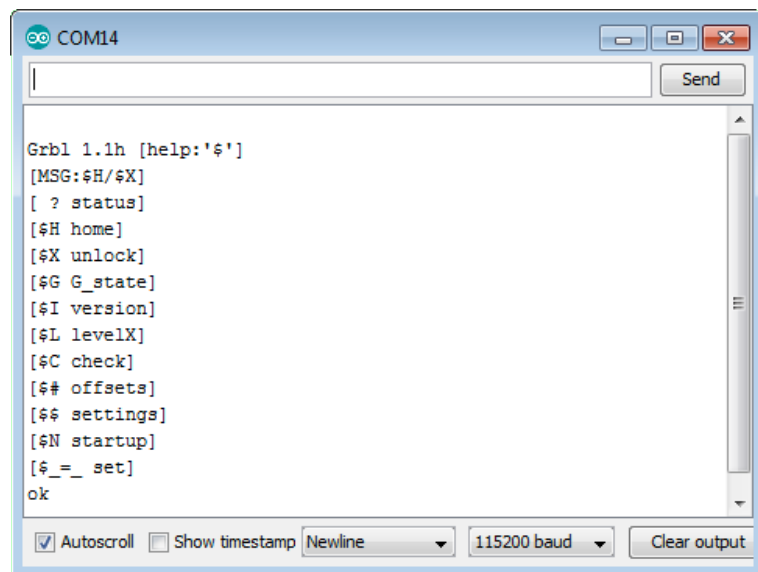


Using Arduino's Serial Window, we're about to send '\$' to Grbl. Grbl's welcome text is shown in the bottom text box.

When we type `>$` and press ENTER, the interface sends that command to Grbl. Grbl then processes the command and sends a response back to the Serial Window (via the bottom text box).

As expected, ('\$') is a simple help menu listing the various tasks Grbl can perform.

**Note:** To save space, future graphics are omitted. Instead, all commands sent to Grbl begin with '>' and are highlighted in gray; responses from Grbl are only highlighted in gray.



### 18.2.1 - Querying Status/Settings

Referencing the help menu, let's send the 'version' command to Grbl:

`>$I` (capital 'i')

**Note:** The '>' is never actually typed. Just type '\$I' (see note above).

**Note:** CRWrite automatically sends '\$I' after opening the connection (see screenshot on previous page).

Grbl responds with a string of various hardware/firmware versions:

`[grbl:1.1h CR:3A PCB:3B VFD:3A YMD:20200307 ]`  
`ok`

These values are useful for troubleshooting, debugging, upgrades, etc. For example, we're running FW 20200307.

**Note:** To decipher this line, see Appendix B: Response Syntax.

**Note:** Grbl responds `ok` after executing each line (except for errors/alarms). Future `ok` text is omitted.

Next, let's send a 'status' command to Grbl:

`>?`

Grbl responds with the current machine status:

`<Alarm|M:0.000,0.000,0.000|B:14,127|L:0|P0Y0|W:0.000,0.000,0.000>`

These values are useful for telling us the current machine position, probe/limit switch status, etc. We'll explore the details later, or you can take a peak at Appendix B: "Response Syntax".

Next, let's look at Grbl's stored settings:

```
>$$
```

Grbl responds with a long list of parameters:

```
$0=10 (stepPulse)
$1=100 (idleDelay)
$2=0 (stepMask)
$3=0 (dirMask)
$4=0 (stepEn)
$5=1 (limLVL)
$6=0 (prbLVL)
$10=127 (statMask)
$11=0.020 (jncDev)
$12=0.002 (arcTol)
$13=0 (Inch)
$20=1 (softLim)
$21=1 (hardLim)
$22=1 (homeEn)
$23=1 (homeDirMask)
$24=30.000 (homeFine)
$25=2000.000 (homeSeek)
$26=1 (homeDelay)
$27=0.500 (homePulloff)
$30=8000 (rpmMax)
$31=0 (rpmMin)
$100=400.000 (x:stp/mm)
$101=400.000 (y:stp/mm)
$102=400.000 (z:stp/mm)
$110=2540.000 (x:mm/min)
$111=3100.000 (y:mm/min)
$112=3100.000 (z:mm/min)
$120=500.000 (x:mm/s^2)
$121=500.000 (y:mm/s^2)
$122=500.000 (z:mm/s^2)
$130=86.500 (x:mm max)
$131=241.500 (y:mm max)
$132=78.500 (z:mm max)
```

Once again, don't worry about understanding the specific details right now (see Appendix G if you're curious). Just remember that Grbl has a list of parameters that control nearly every aspect of Coast Runner's behavior. We can change these values to suit our needs, but in general that is neither necessary nor recommended; Grbl's parameters are properly configured out of the box to work properly with Coast Runner.

**Note:** Changes to these parameters are persistent, even across power cycles.

**Note:** To restore all parameters to default values, type `>$RST=$`.

**Note:** `>$RST=*` will also reset these parameters, but resets other things, too (see Appendix B).

It might seem a bit odd that we're showing you how to access parameters, but then aren't showing you how they work. We have to start somewhere, and introducing these concepts early makes them easier to reference later.

### 18.3 - Manually Moving Axes

Let's figure out how to move Coast Runner's three axes (X/Y/Z). Suppose we want to move the gantry to the middle. Appendix P reveals that we want to move the 'Y' axis.

**Note:** We recommend cutting out the labels in Appendix P and taping them to Coast Runner.

To move the Y axis to the center, we need to find the total Y axis travel, and then divide that value by two (i.e. the center). One way to determine the total travel is to reference Appendix N (Coast Runner's specifications), which shows the Y travel is 241.5 mm. Another method is to ask Grbl to display its machine settings (just as we did above):

```
>$$
```

Grbl lists all the machine parameters (see above), including Y:

```
$131=241.5 (y:mm max) Note: Grbl's entire response isn't reproduced here (see above).
```

Therefore, to center the Y axis, we divide 241.5 mm by two, which is 120.75 mm. Let's try that:

```
>Y-120.75
```

However, Coast Runner doesn't move at all. Instead, Grbl responds:

```
[echo: Y-120.75]
```

```
[MSG: $H disabled]
```

```
error:9
```

**Note:** You will not see this error in CRWrite if you clicked "Home Now" when you first opened the Manual Operation window. To experience the error, relaunch CRWrite and select "Continue without Homing".

After referencing error code 9 in Appendix E, we see that Grbl is telling us we need to home the machine prior to issuing g-code commands. Grbl doesn't know where each axis is located until the machine is homed, and therefore cannot move to the specified machine location. Specifically, the X/Y/Z axes are positioned with stepper motors running in an open-loop configuration; during the homing process, a limit switch on each axis trips at a specific location. From there, the only way Grbl knows where each axis is located is by counting each subsequent step.

---

Let's tell Grbl to home Coast Runner. Both Appendix A and the help menu ('\$') tell us how to home:

```
>$H
```

Coast Runner should have just fully retracted the Spindle, then simultaneously moved the Gantry to the right and the X Table up to the top. If motion did not occur, verify the power cable is connected and the E-stop is disengaged.

---

Now that we've homed Coast Runner, let's try (again) to center the Y axis:

```
>Y-120.75
```

This time Coast Runner moves the Y axis to the center, then Grbl responds:

```
ok
```

**Note:** We've shown this 'ok' message to highlight that the 'ok' occurs after the g-code line completes.

**Note:** If the gantry doesn't move to the middle, type `$RST=#`, then send `Y-120.75` again.

---

Let's determine how Grbl stores the machine's coordinates in 3D space. Let's ask for Grbl's status:

```
>?
```

As before, Grbl sends the machine status. The absolute machine coordinates are indicated by "|M:X,Y,Z|":

```
<idle|M:-86.000,-120.750,-0.500|B:14,127|L:0|P000|W:0.000,0.000,0.000>
```

As you might expect, we can see that the Y axis is located at Y=-120.750.

But what's going on with the X & Z axes? We never moved them, so why aren't they zero? Let's start with Z (we'll deal with X later). Z=-0.500 means the Z axis is positioned -0.5 mm from the point where the Z limit switch tripped during the homing cycle. Tripping a limit switch during normal operation results in a hard limit alarm, which is a fatal condition (until Grbl is reset). Therefore, to prevent hard limit alarms during motion, we need to make sure to never trip the limit switches, hence the homing routine automatically pulls each axis 0.5 mm away from each limit switch.



To better understand how limit switches work, let's intentionally trip Z by moving to the previous trip point:

```
>Z0
```

Moving exactly to the trip point doesn't guaranteed a limit switch will trip, but almost certainly Grbl will respond with:

```
[MSG:Limit Z]
```

```
ALARM:1
```

```
[MSG: Reset to cont]
```

**Note:** If you don't see the above text, try moving Z back down (Z-1) and up (Z0) a few more times. Remember that 'Z0' is precisely the point where the limit switch tripped during the homing cycle; it's possible the switch isn't going to trip again exactly at that point. If Z won't trip, try again with Y (Y0).

Referencing Appendix E, we find that ALARM:1 indicates that a hard limit has occurred.

Now that we're in an alarm state, we quickly find that Grbl no longer responds to any commands:

```
>?
```

```
>Z-10
```

Grbl neither responds nor moves... not even with an 'ok'. For safety reasons, alarm conditions (see Appendix E) always halt further code execution until the special Soft Reset command is sent.

**Note:** Error conditions do not halt Grbl, so no Soft Reset is required after an error occurs.

---

At this point, there are a few ways to make Grbl respond to our commands:

- Create a new session by closing/re-opening the interface (e.g. closing/reopening the Serial Window).
- Create a new session by unplugging/reconnecting the USB cable.
- Create a new session by engaging - then disengaging - the hardware E-stop.
- Send the special "Soft Reset" command ("|").

The simplest method is to send the Soft Reset command:

```
>|
```

**Note:** This is the 'pipe' character ([shift+] \) on a standard US ANSI QWERTY keyboard).

**Note:** The pipe reset command is not supported in CRWrite 5.1.6 and earlier.

Grbl responds with the same welcome message we saw previously:

```
Grbl 1.1h [help:'$']
```

```
[MSG:$H/$X]
```

Per Appendix B, the Soft Reset command is identical to creating a new session with Coast Runner, EXCEPT that after a Soft Reset the previous absolute machine coordinates are preserved (based on the last homing cycle).

**Note:** See "Appendix E - Recover After an Alarm" for a complete Soft Reset description.

**Note:** Absolute machine coordinates are the ONLY volatile information preserved after a Soft Reset (based on the most recent \$H homing command).

---

During normal operation, each axis accelerates from a standstill and decelerates to a stop. However, when an alarm condition occurs, motion is abruptly halted, which can cause the open-loop stepper motors that drive each axis to lose steps. The faster an axis is moving when an alarm occurs, the more likely the stepper is to skip. Therefore, we recommend re-homing when an alarm occurs while axes are in motion. If no axes are in motion when the alarm occurs, then the absolute machine position accuracy is guaranteed after a Soft Reset.

**Note:** Appendix E, column "Position Guaranteed" tells us whether the positional accuracy is guaranteed after each alarm type. For example, positional accuracy is guaranteed after a Soft Limit alarm.

In the previous example, the Z axis was moving when the alarm occurred, and thus the absolute Z position is no longer guaranteed (Z might have lost steps since last homing cycle). Therefore, to ensure accurate tool placement, we know that we should re-home Coast Runner. However, should is not the same as must. Often times during program development, it doesn't matter if the positional accuracy is off slightly... sometimes you'd rather not lose your present physical tool position, and would rather trade convenience for accuracy. Fortunately, Grbl allows us to "unlock" the machine, without homing. Unlocking Grbl simply allows us to move Coast Runner without first homing:

```
>$X
```

Grbl responds with:

```
[MSG:Unlocked]
```

**Note:** If Coast Runner wasn't previously homed during this session, then Grbl treats the initial X/Y/Z location as the machine origin. This can get really weird until you fully understand soft limits. We don't recommend unlocking a machine that hasn't previously been properly homed at some point during a session.

**Note:** Engaging the E-stop places the processor in reset, which ends the current session. Releasing the E-stop allows Grbl to restart into the power-on state. The previous homed position is lost after an E-stop.

**Note:** If Coast Runner crashes while moving (e.g. into the part), then the virtual position Grbl thinks each axis is at might differ greatly from the actual position in physical space. Be careful when unlocking and moving Grbl after a hard crash: until you re-home, Coast Runner could be in a vastly different position than it thinks.

Let's verify that we've retained the machine position, based on the previous homing cycle:

```
>?
```

Grbl responds with something similar to:

```
<idle|M:-86.000,-120.75,-0.003|B:14,127|L:0|P00Z|W:0.000,0.000,0.000>
```

Sure enough, Y is still at -120.75 and X is still at -86.00. Since both Y & X were stationary when the alarm occurred, we know that those positions remain accurate. We also see that the Z axis tripped at **-0.003** mm, which is fairly close to the ideal 0.000 mm trip point (i.e. the position the Z limit switch tripped during the most recent homing routine). Since Z was in motion when the alarm occurred, the Z position isn't guaranteed, but since we didn't crash the machine, we're probably fairly close to this value, and so we'll continue without homing.

---

Before we send our next command, we need to briefly explore how Coast Runner's limit switch hardware circuitry works:

During normal operation, when a limit switch trips (or un-trips), Grbl only initially knows that a switch changed value. Specifically, Grbl doesn't initially know which switch changed, or whether the switch tripped or un-tripped... Grbl only knows that *something* changed. For safety reasons, there isn't enough time to figure out the details, so Grbl immediately halts all motion. After the machine is disabled, Grbl goes back and determines which switch tripped.

**Note:** This is a hardware limitation related to how some microprocessors handle interrupt vectors.

As we haven't moved Z, it makes sense that the Z limit switch is still tripped. In fact, we can verify Z is still tripped by sending ? (as we just did above). The 'Z' in Grbl's response (**P00Z**) indicates that the Z limit switch is tripped.

**Note:** See Appendix B: "Grbl Response Syntax".

With the above in mind, if we now move Z away from the limit switch, then the limit switch state will change (from tripped to not tripped), which will cause another alarm. Let's try that:

```
>Z-50
```

As expected, we "tripped" the Z limit switch, causing Grbl to immediately halt all motion and reply with:

```
[MSG:Limit ]  
ALARM:1
```

[MSG:Reset to cont]

**Note:** Grbl reports `MSG:Limit` (instead of `MSG:Limit Z`). After halting all motion, Grbl went back to see which limit switch tripped. Since none of the switches were tripped after the fact, Grbl dutifully reports that `Limit` tripped. This may seem like a niche corner case, but it can save headaches later.

---

Let's re-home Coast Runner. Remember that we first need to send the Soft Reset (because we're in the alarm state):

```
>|  
>$H
```

**Note:** Hard Limit alarms are disabled during the homing cycle. Therefore, the homing cycle will complete successfully even if one or more limit switches are tripped.

Once again, let's trip the Z axis by moving to the homed zero position:

```
>Z0
```

And once again, Grbl enters the alarm state:

```
[MSG:Limit Z]  
ALARM:1  
[MSG:Reset to cont]
```

Since only the Z axis was in motion when the previous alarm occurred, we can choose to re-home only that axis:

```
>|  
>$HZ
```

**Note:** Homing a single axis enables motion on all axes, even those that were not previously homed.

**Note:** Do not home X (\$HX) without first verifying the Z axis is retracted (e.g. \$HZ). The X Table will crash into the spindle/tool if the Z axis is plunged too far down.

---

Let's revisit the previous question about the X/Y/Z coordinates immediately following a homing cycle:

```
>?  
<Idle|M:-86.000,-0.500,-0.500|B:14,127|L:0|P000>
```

We previously discussed why Y & Z are placed at -0.5 mm after homing (i.e. to prevent limit switch tripping). But what's going on with X? Why is the homed X position -86.000 mm? Before explaining X, let's explore the Y & Z homing routine. While homing, the Y & Z axes move in the positive direction. As each limit switch trips, that position becomes 0.000. We choose to set the Y & Z limit switch trip-point to 0.000 for two reasons:

- CNC machines traditionally operate in negative space (e.g. between -78.500 & 0.000 mm).
- Each axis isn't physically able to travel (too far) beyond its limit switch; all travel is away from the trip-point.

The homed X position (-86.000) is different simply because the X limit switch is physically located at the most negative point on the X axis' travel. Therefore, the X axis homes in the negative direction, and so when the X limit switch trips, the X position is set to the most negative value (-86.500). Grbl then pulls off 0.5 mm (in the positive direction), bringing the X axis to -86.000 mm. Hence, the homed position is -86.000, -0.500, -0.500.

Key takeaway: The Y & Z limit switches will almost certainly trip if we move to absolute machine position Y0 or Z0, whereas the X limit switch will almost certainly trip if we move to X-86.500 mm.

---

### 18.3.1 - Right Hand Rule

Some machinists (incorrectly) believe that Coast Runner doesn't follow the right hand rule ([wikipedia.org/wiki/Right-hand\\_rule](http://wikipedia.org/wiki/Right-hand_rule)). The conceptual hurdle is that the right hand rule is how the tool moves relative to the part. Both the Y & Z axes are easy to conceptualize: when the Y axis moves left (i.e. negative motion), the part 'sees' the spindle is moving left (negative motion). The same is true for Z: when the Z axis moves closer to the part (i.e. negative motion), the part 'sees' the spindle is getting closer. Makes sense, right?

However, the X axis behavior is different:

When the X Table physically moves down (i.e. positive motion), the part 'sees' that the spindle is moving up/away. Imagine you're a plastic army soldier standing on the X table. You only care about where the spindle is in relation to you. Thus, when the table physically moves down, you only see that the spindle is moving up and away from you. It doesn't matter that it's actually the table that's moving... you only see the spindle moving farther away.

Let's try that (while imagining you're standing on the X table):

```
>X0
```

As expected, the X Table just moved down to the bottom, but the toy soldier saw the spindle move up and away.

Now let's move the X Table back to the top (i.e. negative motion):

```
>X-86
```

As expected, the table just moved up to the top, but the toy soldier saw the spindle move down and closer.

Therefore, from the army soldier's perspective, Coast Runner does in fact follow the right hand rule.

As mentioned previously, we suggest taping the coordinate space labels in Appendix P to Coast Runner (as a visual aide).

---

### 18.3.2 - Soft Limits

As mentioned above, Coast Runner only has limit switches on one side of each axis. To prevent crashing each axis into the non-switched side, Grbl includes a "Soft Limit" monitor. When Soft Limits are enabled (default behavior), Grbl verifies that each motion command will not exceed Coast Runner's mechanical limits. Specifically, each axis is bounded between 0.000 and each axis' maximum travel parameter (e.g. 241.5 mm on the Y axis, as previously discussed). If a command would exceed said travel limits, then no motion occurs and Grbl reports an error. Let's test this out:

```
>Y-300
```

As expected, no motion occurs and Grbl responds with:

```
[MSG:Soft Lim]
ALARM:2
[MSG:Reset to cont]
```

It may seem odd that Grbl requires a Soft Reset after a soft limit alarm occurs. The goal is to prevent further motion until the operator acknowledges the unexpected out-of-bounds motion. Fortunately since the axes are stationary when a soft limit alarm occurs, the machine position is guaranteed, so we can soft reset and unlock, then carry on:

```
>|
>$X
```

If you've previously developed G-code using other grbl-based machines, then you've probably learned that you can disable soft limits by typing `$20=0`. When Soft Limits are disabled:

- Grbl does not check position requests against the stored X/Y/Z parameters (\$130/\$131/\$132).
- Soft Limit alarms will not occur. Grbl will attempt to move to any entered position.
- Coast Runner will still halt motion and send an alarm if a limit switch trips
- On the non-limit side, if a motion command results in out-of-bounds positioning, Coast Runner will crash.

We NEVER recommend disabling soft limits, even as a temporary debugging tool. If you understand how both Soft Reset and Soft Limits work, then disabling Soft Limits is almost never required. Disabling soft limits is only required when the current tool position prevents homing, the machine has not been homed this session, and you need to move an axis into positive space... the actual conditions where this might occur are fleeting. Do yourself a favor and keep Soft Limits on (see Appendix B: `$20=1`, or `$RST=#`).

To finish off our introduction to motion, let's move around each axis. First, we'll move the X Table to the bottom:

```
>G0 X0
```

**Note:** *Until you understand machine position, be careful to enter these commands exactly as specified. Otherwise, you could crash the spindle into the X Table, which could mar the spindle and table.*

**Note:** *The example code below is structured to prevent crashes.*

**Note:** Before continuing, verify the X Table is at the bottom.

**Note:** When the X Table is empty, it is impossible to crash into the spindle as long as the absolute X position is between X0 & X-75. Values between X-75 & X-86 can crash into the spindle if Z is down.

**Note:** If a crash occurs, be sure to level the X Table (see 12 - Leveling X Table).

Next let's move the Y axis all the way to the left:

```
>Y-241.5
```

Now let's plunge Z all the way down (towards us):

```
>Z-78.5
```

At this point, each axis is as far from its limit switch as possible. Specifically, the spindle is all the way down, the gantry is all the way left, and the table is all the way down. We can verify machine position by sending:

```
>?
```

Sure enough, the tool is located precisely where we told it to go:

```
...[M:0.000,-241.500,-78.500]...
```

---

### 18.3.3 - Tool Install Coordinates

Let's move to a good position for installing tools:

```
>X0 Y-120 Z-78.5
```

Notice that only the Y axis moved. This is because the X axis was already at X0, and the Z axis was already at Z-78.5. We don't need to install a tool right now, but in the future this position provides easy wrench access.

Next let's move near the homed position:

```
>X-75 Y-1 Z-1
```

All three axes moved simultaneously. Feel free to move X/Y/Z around more; be careful moving X too negative (up).

---

## 18.4 - Modal Parameters

Modal parameters are g-code values that persist until changed by a subsequent value. So far we haven't specified any modal g-code parameters (we've just moved around using the default values). Here's a few modal groups:

Modal Group	Valid Modes (default in bold)	Brief Description
<b>Motion Mode</b>	<b>G0</b> G1 G2 G3 G38.x G80	Defines motion behavior
<b>Distance Mode</b>	<b>G90</b> G91	Defines absolute or relative motion
<b>Unit Mode</b>	G20 <b>G21</b>	Defines units as inches or mm

**Note:** Not all modal groups are shown. See Appendix B for a complete list.

**Note:** Appendix A describes each individual mode (e.g. G0, G1, etc).

**Note:** Modes in **bold** are Grbl's default values after each initialization/reset.

We can view Grbl's complete modal status by typing:

```
>$G
```

Grbl responds with the current modal configuration (a.k.a. "parser state"):

```
[GC:G0 G54 G17 G21 G90 G94 M5 M9 T0 F0 S0]
```

Yikes, that's a lot of modes! For now, let's examine the three bolded modes. Referencing Appendix A:

- G0** (Motion Modal Group): Moves the gantry as fast as possible in a straight line.
- G90** (Distance Modal Group): Moves in absolute coordinate space (based on the last homed position).
- G21** (Unit Modal Group): Tells Grbl to treat all numbers as millimeters.

Suppose we want to slowly move the Y axis to the center. We previously determined that the Y center is approximately Y-120. Appendix A shows that G1 allows us to specify a maximum travel velocity. Let's try:

```
>G1 Y-120
```

Grbl didn't like that:

```
[echo: G1Y-120]
[MSG:G-code F?]
error:22
```

**Note:** This is a syntax error - not an alarm - so we don't need to Soft Reset. Errors occur either when the input syntax is incorrect, or when the command isn't allowed (based on the present machine state).

Referencing error 22 in Appendix E, we see that we need to specify a feed rate (i.e. the maximum velocity):

```
>G1 Y-120 F480
```

Grbl is now moving Y to the middle (Y-120) at 480 mm/minute. From the home position, this will take 15 seconds.

---

Let's check the parser state again:

```
>$G
[GC:G1 G54 G17 G21 G90 G94 M5 M9 T0 F480 S0]
```

Grbl is now in mode G1, and has specified feed rate 480 mm/minute. Until we specifically change these parameters, Grbl will continue to move at 480 mm/minute. Let's test this out by moving X:

```
>X-40
```

Sure enough, the X axis also moves at 480 mm/minute. The key point to remember is that the last modal command sent in any given modal group is retained until another modal command from the same group is sent. That's why we didn't have to specifically send the feedrate (F) or motion mode (G1) again (i.e. we only typed X-40).

---

Let's rapidly move the table to the bottom, Y to the left, and drop Z down:

```
>G0 X0 Y-240 Z-75
```

Let's check the parser state again:

```
[GC:G0 G54 G17 G21 G90 G94 M5 M9 T0 F480 S0]
```

We're once again in G0, so Grbl will move as fast as possible.

The feed rate is still set to F480, so we can switch back to G1 without specifying the feed rate again:

```
>G1 X-75 Y-1 Z-1
```

Even though the feed rate has not changed (F480), each individual axis is now moving slower, such that their combined motion is a straight line moving at 480 mm/min. This motion command will take 32 seconds to complete.

While Coast Runner is still in motion, press the red button to engage the E-stop. The steppers immediately stop. The serial connection remains active, but Grbl no longer responds to commands.

Now rotate the red knob clockwise until it pops back out. Grbl will reset, presenting the familiar welcome message:

```
Grbl 1.1h [help:'$']
```

We're back at square one, as if we just plugged Coast Runner in and connected to it. Before we home, let's take a look at the machine status:

```
>?
<Alarm|M:0.000,0.000,0.000|B:14,128|L:0|0000|W:0.000,0.000,0.000>
```

Notice that since we haven't homed yet, Grbl arbitrarily sets the startup position to [M:0.000,0.000,0.000].

While we can technically unlock the machine right now, there are several caveats:

- Grbl has no idea where each axis is physically located.

- Moving Y or Z negative could crash into the end-of-travel (i.e. away from those limit switches).  
**Note:** *Moving X negative cannot crash X into the end-of-travel, as it will hit the X limit switch first.*
- Attempting to move an axis positive (Y to the right, Z up, or X down) causes a Soft Limit alarm.  
 (because the soft limit system requires each axis to remain between its negative travel limit and 0).

Therefore, since Grbl doesn't know where the axes are located, we need to re-home:

```
>$H
```

### 18.5.1 - Relative Motion

Up to now, all movements we've sent to Grbl are based on Coast Runner's absolute machine position, which itself is based on Coast Runner's physical limit switch locations. However, sometimes it makes more sense to move the tool relative only to its present location. For example, let's say we want to move the Y axis 10 mm left from its present location. One method is to figure out the current absolute position, manually subtract 10 mm and then send that absolute value. However, this method is error prone, cumbersome, and is difficult to automate. Fortunately there's an easier method: we can switch Grbl into relative motion mode (G91):

```
>G91 Y-10
```

Sure enough, the Gantry & Spindle moved 10 mm left.

If we repeat the same command, the Y axis will again move 10 mm left:

```
>Y-10
```

We've now moved the Gantry & Spindle a combined 20 mm left from its starting position. Notice that we didn't send 'G91' in the second line; G91 is a modal command, and so remains active until changed.

As always, we can verify this is true by querying the parser state:

```
>$G
```

As expected, Grbl tells us we're in relative motion mode:

```
[GC:G1 G54 G17 G21 G91 G94 M5 M9 T0 F480 S0]
```

Relative motion (G91) is easy to understand: The specified value (e.g. Y-10) is always how far to move the tool relative to wherever it is now. When we're in relative motion mode (G91), nothing else matters... the motion is always relative to the current position... G91 Z-3 will always move the probe down 3 mm, G91 Y-5 will always move the probe left 5 mm, and G91 X+10 will always move the Table down 10 mm. Easy enough, right?

---

Understanding the difference between absolute (G90) and relative (G91) motion is critically important. Since the fundamental motion mechanism changes based on the mode, sending a motion command formatted for the wrong mode is pretty much always going to be a bad time... there's a huge difference between moving Y 1 mm left (G91 Y-1) and moving Y 1 mm left of the Y limit switch (G90 Y-1).

### 18.5.2 - Absolute Motion

For now absolute motion is easy enough to understand, too. We know that in absolute motion mode (G90), Grbl will always move the tool to an absolute position, based on a specified origin. For example, if we send:

```
>G90 Y-10
```

Coast Runner will move the Y axis 10 mm to the left of the homed machine zero. If we then send the same command again:

```
>Y-10
```

Then no motion occurs, as the Y axis is already positioned at Y = -10 mm (absolute machine position). The key point to remember is that absolute motion is always based on a given origin, not the current tool position.

In a few more pages we're going to introduce (Work Coordinate System) WCS offsets, and then absolute motion is going to get a whole lot more difficult... until you understand WCS.



## 18.6 - Controlling the Spindle

Coast Runner's spindle is a closed loop VFD, which allows the spindle to provide full torque while spinning precisely at any specified RPM. It does so by automatically adjusting the current applied to the spindle motor until it reaches the target speed. Over-protection circuitry protects the spindle; a feature is also included to allow the over-current circuit to automatically reset if triggered.

---

There are two different commands that control the spindle (see Appendix A for complete details):

- "Mn" sets the desired rotation direction. M3 is clockwise, M4 is CCW, and M5 disables the spindle.

**Note:** When Grbl is in mode M5, a safety interlock circuit engages to prevent unintended rotation.

- "Sn" sets the desired RPM, where n is any integer from 0 to 8000 RPM.

**Note:** Values above "S8000" are accepted, but the spindle won't actually spin faster.

**Warning:** When the power & USB cables are connected - and the emergency stop is disengaged - software commands can cause the spindle to rotate at any time. When the above conditions are met, DO NOT TOUCH installed cutting tools except as specifically indicated in CRWrite and/or this manual.

Both 'Mn' & 'Sn' are modal, so their state is always accessible via:

```
>$G  
[GC: G0 G54 G17 G21 G90 G94 M5 M9 T0 F0 S0]
```

As we can see, the default spindle parameters are:

**M5** spindle is disabled  
**S0** RPM set point is zero

---

Let's activate the spindle!

**Caution:** Prior to activation, the collet nut must either be tightened (around a tool) or removed entirely.

To prevent an over-current condition from occurring, use the spindle startup & shutdown sequence outlined below:

**>S0** **Note:** The RPM set point MUST be S0 prior to enabling the spindle.

Most cutting tools are designed for clockwise milling, so:

**>M3** **Note:** Do not switch from clockwise (M3) to CCW (M4) without first setting RPM to zero (i.e. S0).

Even though the spindle is now in "clockwise" mode, the spindle does not rotate because the RPM is zero. Rather than immediately updating the spindle speed to full RPM, we should first start the spindle at a slower speed:

**>S5000** **Warning:** Entering this command will cause the spindle to rotate. Keep hands away from the spindle.

The spindle is now rotating clockwise at a slow RPM. We recommend starting at S5000.

When building this spindle startup routine into a programmatic file, we need to give the spindle time to get up to speed prior to sending the next command. Referencing Appendix A, we find that G4 allows us to specify a delay:

**>G4 P2** **Note:** This two second delay command isn't required when manually typing in commands.

Now that the spindle is rotating, we can (optionally) increase to full RPM:

**>S8000** **Note:** Once the spindle is rotating, we can change to any RPM ('Sn') value.

**>G4 P5** **Note:** This five second delay allows the spindle stabilize at full speed prior to milling.

With the spindle now running at the specified RPM, we can perform milling operations.

### Why is this routine required?

Coast Runner's spindle is capable of pulling more power than the built-in power supply can provide. Therefore, If a stationary spindle is commanded to suddenly spin at full speed, it will attempt to do so as fast as possible, which will engage the over-current protection circuit. Under extreme cases, this could completely disable the high-power rail. If Coast Runner's fans and LEDs turn off during operation, then the most likely cause is an over-current condition.



To recover from an over-current condition, unplug both the USB & power cables for at least ten seconds, then plug both back in (in either order). The LEDs and fan should turn back on. If not, unplug both cables for a minute .

**Note:** Engaging the E-stop can trip the over-current circuit when the spindle is enabled.

**Note:** The previously mentioned firmware update will reduce the over-current likelihood.

When it's time to stop the spindle, we should first set the spindle RPM to zero:

>S0

**Note:** The RPM set point *MUST* be S0 prior to disabling the spindle.

**Note:** Always send 'S0' prior to sending M3/M4/M5.

When building this g-code into a milling file, we need to wait for the spindle to stop:

>G4 P2

With the spindle now stopped, we should send the "disable rotation" command:

>M5

---

Summarizing the above, we recommend the following g-code to start the spindle:

>S0 **Note:** If the spindle was previously stopped correctly, the spindle modal value should already be S0.

>M3

>S5000

>G4 P2

>S8000

>G4 P5

After milling is finished, we recommend the following g-code to stop the spindle:

>S0

>G4 P2

>M5

## 18.7 - Probing with G-code

This section focuses on how to use g-code commands to probe with Coast Runner.

**Note:** See "9 - Probing - Theory of Operation" for a complete hardware description.

Before we probe, let's home Coast Runner and determine the machine position:

>\$H

>?

Grbl returns - among other things, the machine position:

<...[M:-86.000,-0.500,-0.500]...>

---

Grbl supports four probing methods: G38.2, G38.3, G38.4, G38.5 (see Appendix A). In most cases G38.4 & G38.5 are not useful due to Coast Runner's specific probe implementation. G38.2 is BY FAR the most common probing operation, and is the only one we'll use in this primer. G38.2 tells Grbl to move the specified axis until the probe trips. If the probe does not trip after moving the specified distance, then an alarm occurs.

**Note:** G38.3 is identical to G38.2, except that no alarm occurs if the probe never trips. We don't recommend using G38.3 in milling files, since we expect the probe to trip during every probe. Using G38.2 will halt further code execution and alert the user that the probe did not trip, whereas G38.3 will continue processing G-code (almost certainly with invalid probe results).

Before probing, plug the Probe Cable into the Probe Connector, then hold the Probe Cable's ring terminal away from any Coast Runner metal surfaces. Verify the Probe LED is off, then type:

>G91 G38.2 Y-120 F30

Coast Runner is now moving the Y axis up to 120 mm to the left at 30 mm/min, which will take four minutes.

**Note:** To save time, we typically want to be much closer to our work piece prior to probing.

**Note:** Probing accuracy decreases as feedrate increases. We recommend probing at 30 mm/min.

**Note:** G91 moves the probe relative to the current location. G90 is allowed, but not recommended.

At any point over the next four minutes, touch the Probe Cable's ring terminal to the spindle (or other metal surface). The probe cycle immediately stops and Grbl returns the machine coordinates at the point where the probe tripped:

```
[PRB:-86.000,-17.258,-0.500:1]
```

**Note:** The ':1' indicates that the probe successfully tripped.

Since only the Y axis moved during the probe routine, only the Y position has changed (from -0.500 to -17.258).

**Note:** The probe result is ALWAYS shown in absolute machine coordinates (not the present WCS). As we will explore later, this ultimately doesn't matter much, as we tend not to care about the "[PRB:X,Y,Z:1]" result; there's a better way to automatically store probe results into the WCS, which we will explore later.

### 18.7.1 - Common Probe Failures

Let's look at two common probe failures. A probe alarm occurs if the probe is already tripped when G38.2 or G38.3 is called. Let's verify this behavior by touching the Probe Cable's ring terminal to the X Table, then typing:

```
>G38.2 Y-5
```

```
[MSG:probe]
```

```
ALARM:4
```

```
[MSG:Reset to cont]
```

As expected, an alarm occurs because the probe is shorted. Let's Soft Reset to continue:

```
>|
```

Since all axes were stationary when the alarm occurred, machine position is guaranteed after Soft Reset. Therefore we can unlock Grbl (without re-homing):

```
>$X
```

The same alarm will occur if the Probe Cable is not plugged into the Probe Connector. This guarantees that an alarm will occur if you forget to plug in the Probe Cable (instead of crashing the end mill into the work piece).

Let's see what happens if the probe doesn't trip during the probe cycle. Before typing this command, plug the Probe Cable back into the Probe Connector, then verify the Probe Cable's ring terminal isn't touching anything metal in Coast Runner. Do not touch the ring terminal to metal while this command is executing:

```
>G38.2 G91 Y-5 F30
```

**Note:** We've decreased the probe distance to 5 mm, so we don't have to wait as long to see the result.

**Note:** After soft reset, we need to specify all modes again (G91 & F30).

After ten seconds (5 mm divided by 30 mm/min times 60 seconds per minute), Grbl will return the following:

```
[MSG:probe]
```

```
ALARM:5
```

```
[MSG:Reset to cont]
```

If you don't want this alarm to occur when the probe does not trip, use G38.3 instead (see Appendix A).

Once again, let's send the Soft Reset command and then unlock Grbl:

```
>|
```

```
>$X
```

### 18.7.2 - Rotating Spindle While Probing

So far we've probed while the spindle isn't rotating. However, a stationary tool's diameter isn't constant because cutting tools have flutes, thus making them non-cylindrical. For most cutting tools, the difference between the minimum and maximum angular cross-sectional diameters is too large to accurately probe the part.

One solution is to probe the part with a precision cylindrical shaft, which has no cutting flutes. Since the diameter is constant, we can calculate exactly where the spindle is located in relation to the part. However, this requires extra tools, additional tool changes, and isn't actually necessary...

A simpler solution is to probe while the actual cutting tool is installed and rotating. As long as we don't probe towards the part too quickly, we can now assume that the probe contact always occurs at the maximum tool diameter (e.g. 0.250" on a 0.250" tool). Therefore, when probing we recommend configuring the spindle as follows:

>M4	<b>Note:</b> Spin CCW. We recommend rotating the tool the "wrong" way, to prevent cutting.
>S5000	<b>Note:</b> Start the spindle.
>G4 P2	<b>Note:</b> Pause two seconds
>S8000	<b>Note:</b> Spin full speed to warm up the spindle. Not required if spindle already warm.
>G4 P5	<b>Note:</b> Five second warmup pause.
>S5000	<b>Note:</b> Slow spindle back down to probing speed.
>G4 P2	<b>Note:</b> Wait for spindle to reach probing speed.

A few key recommendations for optimal probing:

- The spindle should rotate slowly. Spinning the tool too fast can prevent the probe circuit from tripping, due to a 110 kHz lowpass filter designed to prevent false tripping. Since the tool's OD(max) cutter engagement circumference is typically only a small portion of the overall circumference, we need to make sure the cutting surface contacts the part long enough for the probe to trip. The tool should continuously conduct for at least 50 us, but ideally 100 us or more. Disregarding the math, slower is typically better. We recommend S5000.
- The tool should rotate backwards to minimize cutting. We want to probe the surface without creating chips.  
**Note:** When cutting anodized surfaces, we recommend rotating the "right" way, so that the cutting tool can intentionally cut through the non-conductive oxide layer. See "9.1 - Probing Anodized Parts".
- When developing new code, the 'best' probe settings result in minimal audible cutting noises.  
**Note:** In practice the same code will produce different audible results across multiple runs.
- The slower you move an axis during probing, the more accurate the probe result.

---

### 18.7.3 - Probing Example

When developing probing code for a new work piece, we need to identify a flat face perpendicular to each axis. Probing curved, non-flat, and/or non-perpendicular surfaces is beyond the scope of this manual.

Once we've determined where we're going to probe, we need to move the tool to the Install Position, which positions the tool in such a way that the user can easily install the work piece in the correct location.

For this example, we're going to probe the Y axis in the negative direction (i.e. left). You need to install something metal such that the tool is 2-20 mm to the right of the surface you want to probe. You can install a metal block or whatever else you have handy... just make sure it's electrically isolated from the X Table and connected to the Probe Cable's ring terminal (see "9 - Probing - Theory of Operation").

Before continuing, verify:

- A work piece is securely installed to the X Table.
- The Probe Cable is connected to the work piece and the Probe LED is off.
- The tool is positioned 2-20 mm to the right (we'll move the tool left in the example).

Alright, let's probe!

First we need to configure the spindle to rotate. If you followed the instructions on the last page, the spindle should already be spinning and ready to probe. If the spindle is stopped, go back and follow 18.7.2.

With the spindle spinning, let's send the probe command:

>G38.2 G91 Y-20 F30	<b>Note:</b> If the Y axis continues moving after the tool contacts the probed surface, engage the E-stop.
---------------------	--

This line tells Grbl to move the Y axis up to 20 mm left (Y-20) relative to where it is now (G91) at 30 mm/minute (F30).

When the probe trips, Grbl halts motion and sends the absolute probe trip machine coordinates (as we saw before):

[PRB:-68.500,-77.271,-20.500:1]

Congratulations, you've successfully probed!

To prevent the tool from rubbing against the part, we recommend pulling away from the part after each probe:

```
>G91 G0 Y+1
```

For now, let's also stop the spindle:

```
>S0
```

```
>G4 P2
```

```
>M5
```

In production code, you don't actually need to stop the spindle after each probe. We're just suggesting it so the spindle doesn't drive you crazy while you're reading the next section.

Now what? We need to store the probe result into memory, so that our properly designed g-code can recall this location later. To do this, we need to learn about Work Coordinate System offsets (WCS offsets).

### 18.8 - Work Coordinate System (WCS) Offsets

Earlier we mentioned that the absolute coordinate system (G90) was going to get harder once we introduce Work Coordinate System (WCS) offsets... well here we are. Now might be a good time for a quick break.

WCS offsets are a method to store several different origins simultaneously in memory. Grbl supports six different WCS modes: G54/G55/G56/G57/G58/G59. At any given time, we can only be in one WCS. Since WCS is modal, we can see which WCS modal state we're in by sending:

```
>$G
```

Grbl tells us we're in G54, which is the default WCS:

```
[GC:G1 G54 G17 G21 G91 G94 M5 M9 T0 F480 S0]
```

To change to a different WCS, we just type it in. For example, to switch to WCS G56:

```
>G56
```

We can verify that we're in G56 by once again typing:

```
>$G
```

Sure enough, we're in G56 (whatever that means):

```
[GC:G1 G56 G17 G21 G91 G94 M5 M9 T0 F480 S0]
```

So now we're in G56. Right now this doesn't mean anything, as we still don't know what WCS offsets *do*.

WCS offsets allow us to abstract the actual machine position (which we generally don't care about) from the actual part location (which we do care about). Using Coast Runner's built-in probe, we can accurately determine where the part is located on each axis (i.e. where the probe trips), and then store those probe results in a WCS (we'll explain the actual process later). If we then set our CAM program's origin to the same WCS origin, then our pre-configured g-code will correctly machine the part in 3D space, no matter where the part is physically located in Coast Runner. This is amazingly powerful because it allows us to roughly install the part, and then use the built-in probe to determine exactly where the part is located.

There's a lot to unpack here... we'll circle the wagon in a minute, but for now the key takeaway is: WCS offsets are used to store useful locations in the machine, typically based on a probe result. This allows us to use the same g-code file, no matter where the part is installed in Coast Runner.

---

Why do we need six different WCS offsets? The short answer is we can simultaneously store several different positions in the machine. Some designs might only use one WCS, while others might use three or more. It's generally up to the g-code designer to figure out how many WCS offsets to use.

To view all six stored WCS offset values, send (see Appendix B):

```
>$#
```

Grbl responds with quite a few items (some not shown below), including the XYZ values for all six WCS offsets:

```
[G54:0.000,0.000,0.000]
[G55:-30.000,10.000,9.000]
[G56:0.000,0.000,0.000]
[G57:0.000,0.000,0.000]
[G58:0.000,0.000,0.000]
[G59:0.000,0.000,0.000]
```

If you've previously used this Coast Runner to machine parts, it's likely that G55/G56/G57/G58/G59 will have non-zero values, which is ok. However, since G54 is Grbl's default WCS, the G54 values should always remain 0.000. Otherwise, the machine boundaries defined throughout this manual will differ by that offset, which is a common reason the tool doesn't move where it "should" have moved.

If G54 is not 0.000, then we need to clear the offsets:

```
>$RST=#
```

Grbl responds with:

```
[MSG:Restore:defaults]
ok
```

```
Grbl 1.1h [help:'$']
```

**Note:** \$RST=# zeroes all WCS offsets, but also resets all modal parameters (\$G) to power-on defaults. Remember to re-send all modal parameters after sending \$RST=#. We'll handle this in this demo.

Let's view the WCS offset values again:

```
>$#
```

```
[G55:0.000,0.000,0.000]
```

**Note:** Only G55 is shown; all other WCS offsets have not changed (remain zero).

As expected, all WCS offsets are now zero.

### 18.8.1 - Setting WCS Offsets - Absolute (L2)

Suppose we only want to update one WCS (leaving the others alone). Enter the G10 command, which allows us to store specific axes in specific WCS offsets. There are two different ways to call G10. We'll start with the less common 'L2' method, which sets the specified WCS offset to exactly the entered value (see Appendix A):

```
>G10 L2 P2 X10 Y10 Z10
```

The P2 value tells Grbl to update WCS G55... g-code dates back more than fifty years. Back then, g-code was on punch cards and CNC computers were pretty simple. P1 to P6 correspond to G54 to G59, respectively. As a table:

G10 P Value	Updated WCS	Note
P1	G54	Grbl's default WCS is G54. We recommend NEVER storing any offset in G54, as this will change Coast Runner's default G90 motion behavior.
P2	G55	The remaining WCS offsets are primarily used to store probe results.
P3	G56	
P4	G57	
P5	G58	
P6	G59	

After sending the above WCS update command, let's check the WCS offsets again:

```
>$#
```

Grbl responds with exactly the above offsets in G55:

```
[G55:10.000,10.000,10.000] Note: Only G55 is shown; all other WCS offsets have not changed.
```

In practice, we typically don't use this first 'L2' method, except to clear offsets on a given WCS. For example, instead of sending \$RST=# to clear WCS G55 (as we did above), we can instead send:

```
>G10 L2 P2 X0 Y0 Z0
```

Which clears WCS G55 back to zero, no matter where the gantry is currently located:

```
>$#  
[G55:0.000,0.000,0.000] Note: Only G55 is shown; all other WCS offsets have not changed.
```

Since we don't typically use this method, we're not going to explain the entire concept here. The key takeaway is that you can use this command to set any WCS offsets to zero (without zeroing the other WCS offsets).

---

### 18.8.2 - Setting WCS Offsets - Relative (L20)

Storing relative offsets allows us to save the probe trip point into a WCS origin. In the example above, we probed the Y axis. If you've been following along since 18.7.3, then the tool should still be 1 mm to the right (on the Y axis) from the surface you probed. If not, repeat the steps in 18.7.3.

Since we just probed Y, we need to store the result into a WCS. We'll choose WCS G56, which is linked to P3:

```
>G10 L20 P3 Y0
```

We're now using G10's 'L20' mode, which is a relative offset. Since we only specified Y, only Y is updated.

**Note:** While probing, we typically only update one axis at a time.

Let's see what WCS G56 looks like now:

```
>$#  
[G56:0.000,-76.271,0.000]
```

**Note:** Only G56 is shown; all other WCS offsets have not changed.

**Note:** The stored Y value you see will almost certainly be different, since it's based on your actual probe result from 18.7.3. The key takeaway is that only Y is updated; (X & Z are still zero).

Let's determine the current absolute machine position:

```
>?  
...[M:-68.500,-76.271,-20.500]...
```

As you can see, the Y value stored in WCS G56 is identical to the current absolute machine position.

**Note:** This is only true when we call G10 in relative mode (L20) and the specified axis has a zero value.

---

### 18.8.3 - Accounting for Tool Diameter

In the above example, our stored WCS offset fails to account for two things:

- Tool Diameter.
- The 1 mm we moved Y to the right immediately following the probe.

When designing g-code to machine a part, we typically assign the X/Y/Z origins to physical faces on the part. Therefore, when we probe the part itself, we need to account for the tool diameter. Our example probed Y into the negative direction using a 0.250" tool. Therefore, when Y tripped (at [PRB:-68.500,-77.271,-20.500:1]), the tool was actually 0.125" (0.250"/2) absolutely more positive (i.e. to the right) than the actual Y zero (i.e. the tool center). Therefore, we need to add 0.125" (3.175 mm) to the stored WCS value (so the origin is the tool center point).

We also need to include the 1 mm we moved Y to the right immediately following the probe. Storing WCS values into EEPROM takes a few hundred milliseconds, so we recommend always moving away prior to storing WCS values, so the tool doesn't dwell and rub. Therefore, since we moved 1 mm to the right (positive direction), we need to add 1 mm to the stored position (to account for the post-probe motion).

## Resetting WCS is Advised

Your machine has stored work coordinates from a previous operation. This may affect your current milling operation. It is advised that you clear these values unless you specifically intend this behavior. Would you like CRWrite to reset the work coordinates for you?

YES

NO

Putting both offsets together, we need to add 4.175 mm (3.175+1):

```
>G10 L20 P3 Y4.175
```

To verify we chose our signs correctly, we can retract Z (so we don't crash into the part):

```
>G53 G0 Z-1
```

Here we call G53 to move the tool 1 mm below the Z limit switch. An equally valid method is to retract Z until the tool tip clears the part by a few mm (e.g. G91 Z+10). The goal is to prevent the tool from crashing when we move Y.

Now let's move the tool to Y0, which should center the tool on the probed surface:

```
>G56 G90 Y0
```

If everything worked properly, the tool center should now be coplanar to the surface you probed.

By repeating the above routine on all three axes, we can store the probe results on three separate faces.

Probing takes hands on experience to master. This example should get you familiar with the process, but you'll need to master the art on your own time. Several parting tips/notes that will make your life easier:

- We recommend reviewing the probing routines within existing milling files.
- Z probe results typically don't have a tool diameter offset (the probe contact occurs at the tool center).
- We recommend developing new probing routines with a 1/4" cylindrical spring loosely inserted into the collet (instead of the cutting tool). That way, errant motion simply deflects the spring, rather than crashing the tool into the part. We use McMaster 1986K74.

**Caution:** When using springs, do NOT rotate the spindle at high speed, as the spring will deform.

- For new milling files, we recommend using G53 commands to position the tool prior to the probing routine. This prevents positioning errors on those machines with (incorrectly) stored G54 offsets.

---

### 18.8.4 - WCS Offsets Are Persistent

WCS offsets are persistent, even across power cycles. To be clear, the stored values persist until specifically overwritten or manually reset (as shown above). If at some point in the future you notice that Coast Runner doesn't move an axis to the "correct" location, then the first troubleshooting step is to verify you're in the correct WCS, and then verify G54 has no stored offsets. In a pinch, you can always use G53 to guarantee absolute motion occurs.

---

### 18.8.5 - The WCS Clear Popup

In CRWrite 5.2 a feature was introduced to help with problems that may arise from old WCS values left over from previous jobs interfering with the operation of new jobs. This feature is the WCS Clear Popup.

In all versions of CRWrite after (and including) 5.2, when a user opens a .crproj file to run, a scan is performed to see if WCS registers G54 through G58 contains a nonzero value. If a stored WCS offset is found, the user is shown a popup asking if they want to clear their WCS values.



If the user clicks “Yes” then the WCS registers G54 through G58 are set back to zero. If the user clicks “No”, no clear is performed.

G59 is neither scanned nor cleared by this functionality. This register is intentionally ignored so that, if a .crproj file developer has a use case in which a user must use a WCS value stored in a previous run, they can use G59 for this purpose.

Similarly, a provision was also made for if a .crproj file author does *not* want a user to have the option to clear the WCS values before running their .crproj file. By including a special value in the .crproj file header, a .crproj file author can ensure that the WCS clear popup will not appear when a user runs their .crproj file. This special value is detailed in the “Advanced manifest.yml Features” section of this guide.

## 19 - Creating .crproj Files

The .crproj file format simplifies part sharing amongst users. A .crproj file is simply a .zip archive with the file extension changed (for user clarity). A special "manifest.yml" file - contained at the root directory - defines how the file contents are displayed to the user, via a series of step-by-step instructions. A properly designed .crproj file houses all manufacturing files and user instructions required to make the contained part:

Housed File Type	Purpose	Required Format
Pictures	Display Steps Visually	.BMP, .JPG
G-Code	Automated machine cutting code	Any ASCII text (extension irrelevant)
3D Printable Jigs	Create jigs, if needed	.STL is most common (extension irrelevant)
Part Model	Allow user to modify CAD/CAM	Any model file (extension irrelevant)
Guide Files	Supplemental user instructions	Any file (typically PDF)
Manifest	Defines how files are presented	manifest.yml (must be lower case)

We recommend starting new .crproj designs by unzipping an existing .crproj file (e.g. Turners-Cube-v3.crproj), and using its structure.

To create a .crproj file, compress the above items into a .zip container, then change the file extension from '.zip' to '.crproj'.

**Note:** The manifest.yml file *MUST* be at the root level. If you compress the folder containing these items, then the compressed file's root directory will *ONLY* contain that folder. If you then attempt to run this .crproj file in CRWrite, CRWrite will report an error. To prevent this error when creating the .zip archive, select the actual files themselves (i.e. select the files and sub-folders, not the entire root folder).

**Note:** After selecting a .crproj file, CRWrite will verify that all file paths written in the manifest.yml file are valid. If you get an error that the file isn't valid, verify all paths are correctly mapped.

**Note:** Folder and file names within .crproj containers are cAseSeNsItIve.

**Note:** Folders within .crproj files are accessed with forward slashes: MyCodeFolder/MyAwesomeFile.txt

**Tip:** To save time when testing new .crproj files, comment out 'step\_gcode' commands from manifest.yml. This allows you to verify the text and images display as intended, without actually milling the part. Another option is to create a gcode\_test.txt file with no motion commands.

### 19.1 - Creating the manifest.yml File

A root level "manifest.yml" file defines how CRWrite presents the contents - stored inside the .crproj file - to the user. The YAML file is editable in any standard text editor, hence experienced machinists can create .crproj files without a programming background. We recommend the open source Notepad++, which has a built-in YAML visualizer.

**Note:** For the purposes of this document, the terms "YAML" and "yaml" are identical.

Historically "yaml" is the file extensions, whereas "YAML" is the standard.

The manifest.yml files contains the following building blocks:

Syntax*	Description	Example (user text)**
"- job_name: "	A 'job' contains all information required to create a part. Most .crproj files make a single part, and thus contain a single 'job'.	- job_name: Turner's Cube #At least one job_step
"job_text: "	Additional text shown after opening a .crproj file, used to describe each job to the operator.	job_text: Manufacture a Turner's Cube

The manifest.yml files contains the following building blocks (continued):

Syntax*	Description	Example (user text)**
"model_files: "	(optional) One or more files used to: -Create a 3D printed jig -Model the part If included, the user can optionally save all files to a user-specified directory, then 3D print and/or modify.	model_files: - stl/left_jig.stl - stl/right_jig.stl - stl/part.ipt
"guide_files: "	(optional) One or more files used to provide supplementary information. If called, the user can optionally save all files to a user-specified directory, then access directly.	guide_files: PDF/build.pdf
"job_steps: "	Defines the steps to show the user for each job.	job_steps: - step_name: Verify Empty step_text: Verify Coast Runner is empty, then press next to Home. step_image: IMAGE/1-Empty.bmp step_gcode: Code/01_Home.txt - step_name: Install Jig step_text: Install the jig as shown. step_image: IMAGE/2-JigIn.bmp step_gcode: Code/02_probe.txt
"step_name: "	Title text summarizing the action performed in this step.	- step name: Verify Empty
"step_text: "	Detailed instructions for this step.	step_text: My detailed description.
"step_image: "	(optional) Path to image file displayed to user. CRWrite supports the use of both static images and animated gifs.	step_image: IMAGE/4E-Hole.bmp
"step_gcode: "	(optional) Path to g-code file that executes AFTER the user sees 'step_image'/'step_text' (by pressing next).	step_gcode: Code/04_DrillHole.txt
"timeout: "	(optional, step must also contain 'step_gcode') Override default seconds CRWrite waits before returning an error. Timer resets after each Grbl 'ok'.	timeout: 80
"reset: "	(optional, step must also contain 'step_gcode') After the step completes, Grbl: -False: doesn't reset (default) -True: resets	reset: true
"pause: "	(optional, step must also contain 'step_gcode') When 'step_gcode' finishes executing: -False: CRWrite moves to next step (default) -True: User must press 'next' button.	pause: true
"manifest_file: "	(optional) Path to another manifest file. CRWrite will replace this line with the entire contents of the specified file.	manifest_file: mySubmanifest.yml

**Note:** We recommend starting with an existing manifest.yml file

**\*Note:** Trailing spaces are important (see Appendix F - YAML Formatting)

**\*\*Note:** Examples assume the .cproj file has root level folders 'stl' 'PDF' 'Code' & 'IMAGE'

### 19.1.1 - Example manifest.yml File

- job\_name: Beer Bottle Opener  
job\_text: Manufacture an 80% beer bottle opener from raw stock.  
job\_steps:
    - step\_name: Verify Empty  
step\_text: Is mill Empty? Verify nothing is installed.  
step\_image: 01\_Empty.jpg  
step\_gcode: 01\_Home.nc
    - step\_name: Step2 #This step doesn't show a picture or execute code  
step\_text: This step only displays this text (it's so meta).
  - guide\_files: #File save prompt at job beginning regardless of order.
    - exampleGuide1.pdf
  - model\_files: #File save prompt at job beginning regardless of order.
    - exampleModel1.stl
  - job\_name: Beer Bottle Breaker  
job\_text: Manufacture a beer bottle breaker from raw stock.  
job\_steps:
    - manifest\_file: BeerBottleBreaker.yml
- 

## 19.2 - Advanced manifest.yml Features

In addition to the manifest.yml syntax defined above in section 19.1, there are several other “advanced” features that you can use in manifest.yml to enrich the functionality and presentation of your .crproj file. These are defined below.

*Note that in general these features are available only in CRWrite 5.2 or beyond. These will not work, or will not work as expected, in earlier versions of CRWrite.*

### 19.2.1 - Using Markdown Formatting

The default step\_text feature only allows you to output unformatted plain text in your .crproj files. If you would like to add formatting, including line breaks, headings, bold and italics, you can instead use “step\_markdown”.

step\_markdown allows you to include special symbols in your plaintext to define how it should be formatted. If “step\_markdown” is included in a job step, then CRWrite will pass the value of the step\_markdown field through a Markdown processor and will then show the output in the step body. Note that if “step\_markdown” is defined in a job step, then any value in “step\_text” will be ignored.

The Markdown implementation that CRWrite uses is defined at <https://daringfireball.net/projects/markdown>. A convenient guide to the Markdown language can be found at <https://markdownguide.org>.

Here is an example of how to include formatted text using step\_markdown

```
- job_name: Sample Markdown Job
  job_text: This job will show how markdown can be used to format CRWrite
description text.
  job_steps:
    - step_name: Markdown Example
      step_markdown: |
## Markdown Heading
Using markdown I can output normal, italic, or bold text.

I can also introduce line breaks to help organize my text.
      step_text: You should include a "step_text" value as a fallback
```

The description text for this step will output as:

#### Markdown Example

Using markdown I can output normal, *italic* or **bold** text.

I can also introduce line breaks to help organize my text.

Note that when using `step_markdown` you should also include a `step_text` value in the same job step. This is so that if your `.crproj` file is opened using an older version of CRWrite which does not support `step_markdown` some description value will still be displayed for your step.

### 19.2.2 - Hiding the WCS Clear Popup

Section 18.8.5 includes describes functionality in CRWrite 5.2 and beyond which asks users to clear local WCS registers before running a `.crproj` file. This functionality is enabled by default, but some may not be appropriate for some `.crproj` files. In these cases, you can insert a special value into your `manifest.yml` to prevent the “Clear WCS” popup from appearing when running that `.crproj` file.

To hide the WCS Clear popup, insert the key / value pair “`disable_wcs_clear_prompt: true`” into the top of your `manifest.yml`. See the box below for an example of this:

```
- job_name: Example - Don't Clear WCS
  job_text: This job will show how to hide the "Clear WCS" popup.
  disable_wcs_clear_prompt: true

  job_steps:
    - step_name: Don't Clear WCS Example
      step_text: Note the key / value pair in the top of this file
```

### 19.2.3 - Using Submanifest Files

There are some cases in which a certain step or set of steps may be commonly repeated in a `.crproj` file. For example, perhaps your `.crproj` file contains five different jobs, and all of those jobs start with the same tool install sequence. In these cases, to ease maintenance and reduce file size, you can use a “submanifest” to define your set of steps once and then import it where needed.

To import a submanifest file, simply insert a step line that says “`manifest_file: <manifest_file_name.yml>`” into your set of job steps. The submanifest file itself should be formatted similarly to a normal manifest file: it should be a set of job steps. *Note: the submanifest file should not contain a header, `job_name`, `job_text`, etc.*

This might look something like this:

#### Parent manifest.yml

```
- job_name: Submanifest Example - Parent
  job_text: This manifest.yml file contains a submanifest import

  job_steps:
    - step_name: Sample step 1
      step_text: This is a normal step
      step_gcode: Code/step_1.gcode

    - step_name: Sample step 2
      step_text: Another normal step

    - manifest_file: submanifest.yml

    - step_name: Sample step 3
      step_text: The manifest file continues on as normal
      step_gcode: Code/step_2.gcode
```

## Child submanifest.yml

```
- step_name: Submanifest Step 1
  step_text: Normal step inside of a sub manifest

- step_name: Submanifest Step 2
  step_text: Another sub manifest step

- step_name: Submanifest step 3
  step_text: Submanifests can contain gcode
  step_gcode: Code/step_2.gcode
```

A submanifest is essentially a macro. Whatever steps you define in the submanifest will be inserted into the parent manifest at exactly the location of the import statement. The submanifest can contain steps formatted just like any step inside the parent manifest.yml file. You can also import submanifests into other submanifests.

### 19.2.4 - Enforcing a Minimum Software Version

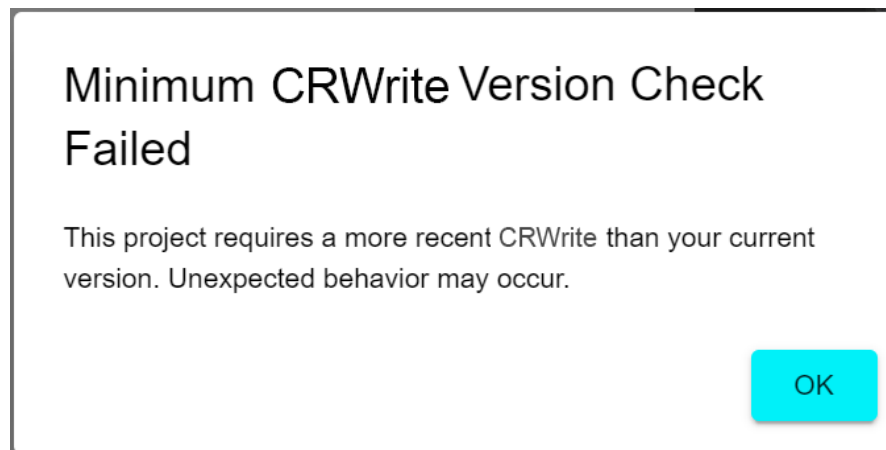
If your .crproj file utilizes certain new CRWrite features, you may wish to warn users if they attempt to run the software on an earlier version of CRWrite. You can do this by adding a line to manifest.yml which will show users a warning popup if they attempt to run your .crproj file on an older version of the CRWrite software.

To set a minimum software version, insert the key / value pair “min\_software\_version: <version\_number>” into the top of your manifest.yml. See the box below for an example of this:

```
- job_name: Example - Minimum CRWrite Version
  job_text: This job will show a warning if opened on a version of CRWrite older
  than 5.2.2.
  min_software_version: 5.2.2

  job_steps:
    - step_name: Minimum CRWrite Version Example
```

A user trying to run your .crproj file on an earlier version of CRWrite will receive the following warning popup:



Note that the user will not actually be *prevented* from running your code - they will simply be warned about it.

Note also that this feature was introduced in CRWrite 5.2, and as such will only work on this or later versions of CRWrite 5.2 - meaning that if a user tries to run your code on, say, CRWrite 5.1, they won't receive the warning popup, because the popup didn't exist in CRWrite 5.1.

### 19.2.5 - Enforcing a Minimum Firmware Version

If your .crproj file utilizes certain newer firmware features, you may wish to warn users if they attempt to run the cutcode on an earlier 328p firmware version. You can do this by adding a line to manifest.yml which will show users a warning popup if they attempt to run your .crproj file on an older firmware version.

To set a minimum firmware version, insert the key / value pair “min\_fw\_version: <version\_number>” into the top of your manifest.yml. See the box below for an example of this:

```
- job_name: Example - Minimum Firmware Version
  job_text: This job will show a warning if opened on a mill with a 328p
firmware older than 20210609
    min_fw_version: 20210609

  job_steps:
    - step_name: Minimum Firmware Version Example
```

A user trying to run your .crproj file with an earlier firmware version will receive a warning popup. Note that the user will not actually be *prevented* from running your code - they will simply be warned about it.

Note also that this feature was introduced in CRWrite 5.2, and as such will only work on this or later versions of CRWrite 5.2 - meaning that if a user tries to run your code on, say, CRWrite 5.1, they won't receive the warning popup, because the popup didn't exist in CRWrite 5.1.

### 19.2.6 - Enforcing a WCS Offset for a File

There are some cases where a .crproj job may require that the user run it with a certain WCS offset in place. In these cases, you can set up your job to check for this WCS offset and to warn the user with a custom popup if the actual offset does not match the expected one. This can be done using the key / value pairs “wcs\_value\_check” and “wcs\_check\_failed\_message”:

```
- job_name: Example - WCS Offset Check
  job_text: This job will show a error popup and terminate if the G54 offset
does not match the expected values
  wcs_value_check: G54 X1 Y2 Z-3
  wcs_check_failed_message: "Error: WCS in G54 is not correct"

  job_steps:
    - step_name: WCS Offset Check Example
      step_text: WCS must match for this job to run
```

The “wcs\_value\_check” field must start with a WCS offset key - G54 - G59 - and is followed by a grbl-formatted X,Y,Z value. The specified WCS value will be checked to see if the offset values match the X,Y,Z value. If it does not match exactly, the wcs\_check\_failed\_message will be displayed.

### 19.2.7 - Making a Step Unskippable

CRWrite contains a “skip to milling step” button that allows users to skip over purely informative steps. This is useful for experienced machinists who may not need to go through, say, detailed steps on how to install a new tool each time they run code. However, there are some cases in which a crproj file author may wish to convey important information in an informative step that they don't want a user to be able to skip over.

In these cases, you can use the “unskippable: true” tag to ensure that a user cannot skip over a step.

Note that any step with a “step\_gcode” setting is by default unskippable.

This functionality was introduced in CRWrite 5.3.0.



```

- job_name: Example - Unskippable Step
  job_text: Example showing unstopppable steps

  job_steps:
    - step_name: First Step
      step_text: This is the first step in the file

    - step_name: Second Step
      step_text: A user clicking "skip to next milling step" on Step 1 will
                  be able to skip over this step.

    - step_name: Unskippable Step
      unskippable: true
      step_text: A user clicking "skip to next milling step" on steps 1 or 2
                  will skip to this step. This step won't be skipped over.

```

### 19.2.8 - Jumps

By default, clicking "next" in CRWrite will move the user on to the next step. However, you can also create a "jump" to a different step using the "step\_goto" tag like so:

```

- job_name: Example - Jump
  job_text: Example showing a jump
  job_steps:
    - step_name: First Step
      step_text: This is the first step in the file

    ...

    - step_name: Jump Step
      step_text: Clicking next will jump to step 5
      step_goto: 5

```

The address provided to "step\_goto" - in this example, 5, is a zero-indexed, absolute step number. 5 in this example is the 6th step in the overall CRWrite file. Note that this global numbering includes submanifests.

When the user clicks "next" on the jump step, any other pertinent actions (such as running gcode, etc.) will first be performed, and then CRWrite will jump to the specified step. Jumps can be performed both ahead and backwards.

This functionality was introduced in CRWrite 5.3.0

### 19.2.8 - Popups

CRWrite 5.3.0 introduced the ability to display a popup upon clicking "next" on a step. This can be done like so:

```

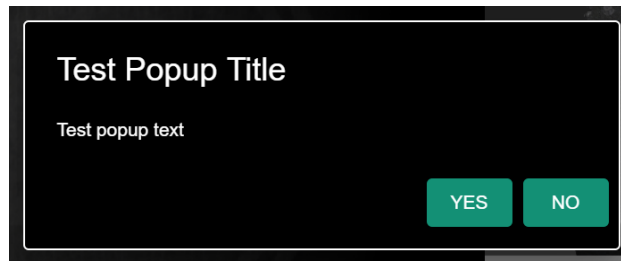
- job_name: Example - Popup Step
  job_text: Example showing a popup

  job_steps:
    - step_name: Popup Step
      step_text: Upon clicking "next" users will see a popup.
      popup_text: The body of the popup
      popup_title: The (optional) title of the popup

```

The displayed popup will have both a "Yes" and a "No" button. By default clicking either button will close the popup and proceed to the next step.

The displayed popup will look like so:



Some useful notes on popups:

- The popup will be displayed only *after* clicking “next”.
- If no popup\_title is provided, no title will be shown (i.e. there is no “default title”).
- If a step contains both “step\_gcode” and a popup, the popup will not be displayed.
- Including “step\_goto” with a popup will perform the correct jump after closing the popup

### 19.2.9 - Popups and Jumps

CRWrite 5.3.0 also introduced the ability to perform a jump using a popup.

The “Yes” and “No” buttons can be overridden such that, when a user clicks either button, they will be jumped to a different step. The jump addressing scheme is the same as with step\_goto: zero-indexed, absolute addressing.

The popup buttons can be turned into jumps using “popup\_yes\_step” and “popup\_no\_step”. Both are optional and each can be used without the other.

```
- job_name: Example - Popup with Jump
  job_text: Example showing a popup with jump

  job_steps:
    - step_name: Popup Step
      step_text: Upon clicking “next” users will see a popup.
      popup_text: The body of the popup
      popup_title: The (optional) title of the popup
      popup_yes_step: 10
      popup_no_step: 5

- job_name: Example - Popup with Jump 2
  job_text: Another example showing a popup with jump

  job_steps:
    - step_name: Popup Step
      step_text: Upon clicking “next” users will see a popup.
      popup_text: The body of the popup
      popup_title: The (optional) title of the popup
      popup_no_step: 7
```

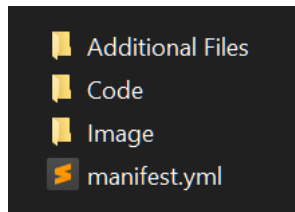
In the first example, clicking “Yes” will jump the user to step 10 (zero-indexed, the 11th step) and clicking “no” will jump them to step 5. In the second example, clicking “Yes” will take the user to the next step like normal, but clicking “no” will jump them to step 7.

---

### 19.3 - Including Additional Files for a User to Extract

Earlier versions of CRWrite provided the ability for .crproj file authors to include files (such as stls for printable jigs) with their .crproj file archives. End users could extract these files using the CRWrite software. As of CRWrite 5.2 we have restored this functionality.

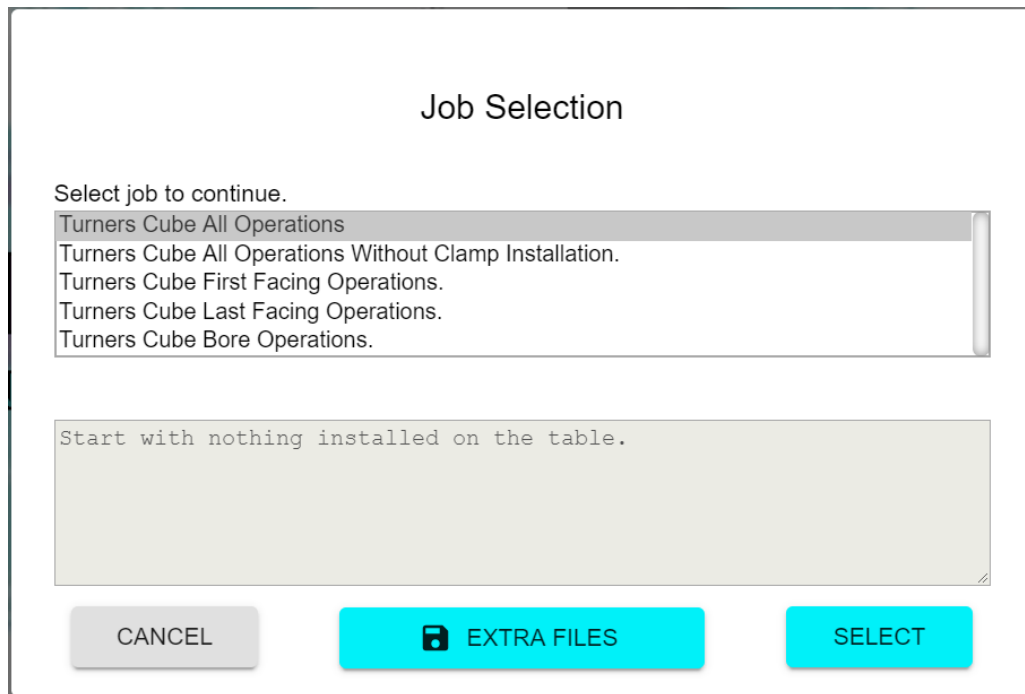
To include additional files in your .crproj package, simply create a folder called “Additional Files” at the root level of your .crproj archive. Then, place whatever files you want the user to be able to extract in this folder.



*Add the “Additional Files” folder to your .crproj archive and zip it up with the other folders / files.*

When an end user loads a .crproj file containing additional files in CRWrite 5.2 or later, they will see an “Extra Files” button on the job selection screen. Clicking this button will open a file dialog in which they can choose where to extract the files.

Note that a user running a version of CRWrite earlier than 5.2 can still run a .crproj file with Additional Files included, but they will not be able to extract the included files.



*The “Extra Files” button that will appear when loading a .crproj file with an Additional Files folder*

---

## 19.4 - Creating G-Code

CRWrite doesn't create g-code... it just runs pre-written .crproj files. Creating g-code is up to you, your brain, and your ability to generate CAM files using a design created in a 3rd party CAD program. Circa 2020, AutoDesk Fusion 360 is the best 'free' program. Even after its 'free' feature set was neutered 2020Q3, no other free program even comes close. We'd love to recommend an open source CAD/CAM solution... but we're still looking for one even remotely comparable to F360. We prefer Inventor, although AutoDesk's SaaS license post-2017 is gross.

Most free CAM programs generate g-code that moves the cutting tool row-by-row across the part, raising and lowering the Z axis as needed, but otherwise paying no heed to actual part geometry. These programs are suitable for cutting wood and plastic, but do not fare well when cutting metal. We recommend generating g-code only with CAM programs that cut along part geometry with uniform radial engagement.

Most aluminum cutting operations should occur at full speed (S8000), whereas steel is typically machined at lower RPMs. For steel cutting operations, we recommend adding a note to your first 'step\_text' entry that tells the operator to verify they have the latest firmware installed (to ensure full torque at low RPM).

We recommend the following starting parameters to cut aluminum:

Mill (climb)		Chip Load (IPT) @ Cutter Dia:			Depth of Cut	
RPM ('Sn')	Method	0.0625	0.125	0.250	Radial	Axial
8000	Slotting	0.0005	0.0007	0.001	1.00 x Dia	0.10 x Dia
8000	Roughing	0.0005	0.0007	0.001	0.60 x Dia	0.25 x Dia
8000	Finishing	0.0005	0.0007	0.001	0.25 x Dia	1.00 x Dia
8000	Adaptive	0.0005	0.0007	0.001	0.04 x Dia	2.50 x Dia

Chip Load (IPT) @ Cutter Dia:

Helical Plunge	0.0625	0.125	0.250
Ramp Angle (degree)	1	1	1
Ramp Diameter (inch)	0.0593	0.1187	0.2375

Chip Load (IPT) @ Cutter Dia:

Drill	0.0625	0.125	0.250
RPM	8000	8000	8000
Feed per Revolution (inch)	0.0593	0.1187	0.2375
Plunge Rate (IPM)	7.2	14.4	24
Peck Depth	0.25 x Dia	0.25 x Dia	0.25 x Dia

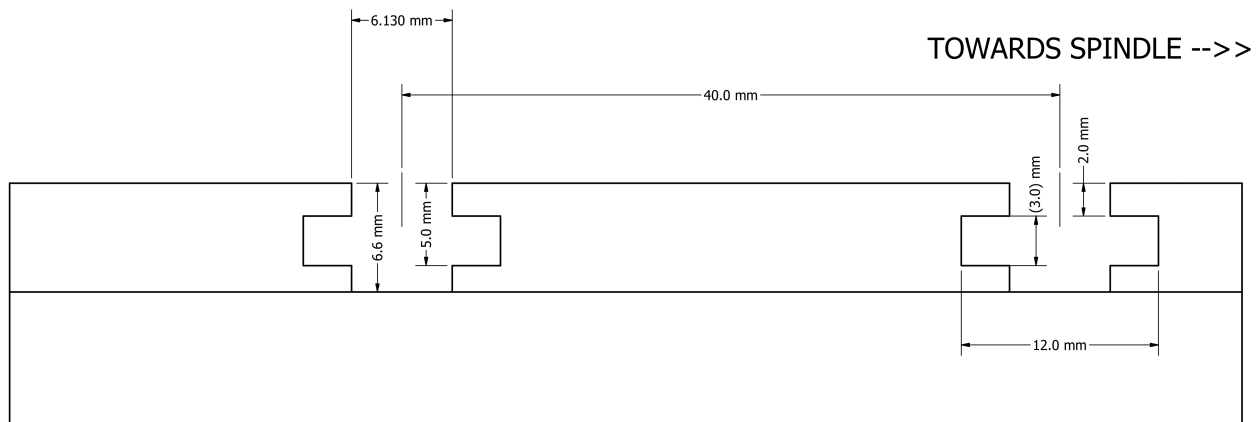
## 19.5 - Designing Custom Jigs

Distributing subtractively-manufactured designs is traditionally difficult due to the various fixtures, clamping tools, and/or vices required for each part orientation. 3D printing attempts to overcome these obstacles by enabling each user to reliably reproduce the exact jig required to secure a given part, even if that part isn't rectangular. 3D printed jigs abstract alignment concepts from the user, enabling automated machine alignment to the part.

3D printed jigs must hold the work piece rigid to prevent cutting vibration. Plastics are weak when tensioned, torqued, or sheared, but remain rigid when compressed (unless they melt). Thus, 3DP jigs should attempt to contact as much surface area as possible between the part and X Table. Apply compression with bolts, but make sure neither the bolts nor the T-Slot contact the part, or else the probe won't work (see 9 - Probing).

### 19.5 - X Table T-Slot Geometry

Custom jigs should have a 6.0 x 1.5 mm boss on the bottom to help square the work piece to the tool. The jig boss rests in either of the X Table's two T-slot profiles, which are spaced 40 mm apart. The T-slot rail dimensions are compatible with industry standard 16 & 20 mm long M4 T-nuts.



*Profile view of X Table, showing T-Slot Profile geometry.*

To reduce chatter, we recommend mounting the work piece to the X Table with at least three T-Slot nuts. Two mounting points is acceptable if the jig has sufficient surface area to the X Table. One mounting point is not recommended, as the jig can easily walk (even with a squaring boss).

# Appendix A: Supported G-Code Commands

Coast Runner supports the following g-code commands (less frequently used commands in gray).

G-Code	Example	Name	Summary
<i>F<sub>n</sub></i>	F10	Feed Rate	Set maximum velocity for speed-limited commands. Unit: G20/G21
G0	G0 X-2 Y-1	Rapid Move	Move as fast as possible in a straight line to the specified point
G1	G1 X-2 F100	Linear Move	Move at specified Feed Rate in a straight line to the specified point
G2	<a href="#">search online</a>	CW Arc	Arc at specified Feed Rate in specified plane.
G3	<a href="#">search online</a>	CCW Arc	Arc at specified Feed Rate in specified plane.
G4	G4 P2.7	Dwell	Pause for specified period. Unit = seconds
G10 L2	G10 P2 L2 X1	Set WCS (absolute)	P1:P6 indicates which WCS - G54:G59 - to modify, respectively. Set specified WCS axis/axes to specified value/values based on machine absolute origin, without movement (current position ignored). Example sets WCS G55's X axis 1 unit from machine's absolute zero.
G10 L20	G10 P2 L20 X1	Set WCS (relative)	P1:P6 indicates which WCS - G54:G59 - to modify, respectively (i.e. P1 is G54, P2 is G55, etc). Set specified WCS axis/axes so current position becomes specified value/values, without movement. Use to set WCS based on probe result. Example sets current X position to 1 unit* in G55. <b>Note:</b> Unit depends on G20/G21.
G17	G17	Set XY Plane	Sets arc plane to XY
G18	G18	Set XZ Plane	Sets arc plane to XZ
G19	G19	Set YZ Plane	Sets arc plane to YZ
G20	G20	Unit = inch	Sets unit to inch.
G21	G21	Unit = mm	Sets unit to mm (default).
G28	G28	Return directly to reference	Return to the last machine position stored by G28.1, in a straight line from present location. If there are no axis/axes (X/Y/Z) on the same line, then G28 will always move to the absolute machine position stored by previous G28.1 command, disregarding the current WCS (G54:G59) and movement mode (G90/G91).
	G28 G91 X0 Z1	Return indirectly to reference	First move relative to present position, as specified by axis/axes. Then move ONLY specified axis/axes to last stored G28.1 position. Example code moves Z 1mm up, then moves ONLY X & Z to previously stored G28.1 absolute machine position. The Y axis never moves in the example code. Note: The described behavior is only true when in relative movement mode (G91). The behavior is entirely different in absolute movement mode (G90), which is generally not recommended unless you are very familiar with G28.
G28.1	G28.1	Store present position	Store the current absolute machine position, for use by G28. This value is persistent, even across power cycles, until overwritten (by another G28.1) or zeroed (see '\$RST=#').
G30	see G28	Same as G28	G30 behavior is identical to G28, except that G30 references G30.1.
G30.1	see G28.1	Same as G28.1	G30.1 behavior is identical to G28.1, except position used by G30.

G-Code	Example	Name	Summary
G38.2	G38.2 X-2 F5	Probe until contact, alarm if no contact	Move up to specified distance at specified feedrate. Motion stops immediately if tool contacts part, and Grbl responds with absolute position. ALARM:5 occurs if tool doesn't contact within specified distance. ALARM:4 occurs if probe is already tripped when called.
G38.3	G38.3 X-2 F5	Probe until contact or distance.	Similar to G38.2, except no alarm occurs if probe doesn't trip after traveling specified distance. Instead, Grbl responds with absolute position, with a trailing :0 (e.g. [PRB:-80.000,-0.500,-0.500:0]) Tip: During development, use in place of G0 for manual non-contact rapids around parts (e.g. at F3100); errant tool contact halts motion.
G38.4	G38.4 X-2 F5	Probe until clear, alarm if does not clear	Move up to specified distance at specified feedrate. Motion stops immediately when tool no longer contacts part. When motion stops - either the tool no longer contacts the part, or the specified travel is reached - Grbl responds with absolute position (in present WCS). ALARM:5 occurs if probe does not un-trip during motion. ALARM:4 occurs if probe is NOT already tripped when called. Note: Unlikely to work if spindle is rotating.
G38.5	G38.5 X-2 F5	Probe until clear, no alarm	Same as G38.4, except no alarm occurs if probe doesn't un-trip.
G43.1	G43.1 Z-.1	Tool Length Offset	Specify a tool length offset. After each tool change, we recommend probing Z (e.g. G38.2) instead of using G43.1.
G49	G49	Clear TLO	Sets Tool Length Offset to [0.000]
G53	G53 G0 X-2	Move using machine origin	Move the specified axis/axes to absolute machine position, regardless of active WCS (G54:G59) or movement mode (G90/G91). G53 only applies to g-code on the same line (not modal).
G54 G55 G56 G57 G58 G59	G55 X-2	Work Offset 'WCS'	Work offsets are typically used to define the part origin in relation to probe results. After probing, use G10 L20 ___ to define each WCS offset. Example moves X axis -2 mm from G55 X0. <b>Note:</b> WCS offsets are persistent, even across power cycles. To zero all offsets, see Appendix B: "\$RST=#".
G80	G80	Block Motion	Prevents linear motion until next G0/G1/G2/G3/G38.x is sent.
G90	G90 X0	Absolute Move	Move to specified <u>absolute</u> location in current WCS. Example moves X axis to zero in current WCS.
G91	G91 X0	Relative Move	Move specified distance, <u>relative</u> to current position. Example moves X axis zero units (i.e. no movement occurs).
G92	G92 X0Y0Z1	Coordinate Offset	Source of epic misery! Unless you know what you're doing, use WCS instead. G92 Coordinate Offset is applied to all other offsets (G54:G59 & G28.1/G30.1), without movement. Unspecified axes aren't modified. Not persistent (cleared on reset).
G92.1	G92.1	Clear Offset	Clear previously set G92 coordinate offset.
G93	G93	Minutes/Unit	F interpreted as inverse feed rate.
G94	G94	Units/Minute	F interpreted as feed rate (default).
Nn	N100	Line Number	g-code line number. Maximum allowed value is (2 <sup>31</sup> -1).



G-Code	Example	Name	Summary
<i>Sn</i>	S5000	Spindle RPM	Set spindle revolutions per minute. Values above S8000 are interpreted as S8000. See "spindle operation" for sequencing info.
M0	M0	Pause Program	Halts further g-code execution until the 'Resume' command (~) is sent. Not useful within prebuilt files, but sometimes used during debug.
M2	M2	End Program	Changes ONLY the following modal states to the indicated values: [GC:G1 G54 G17 *** G90 G94 M5 *** *** ***] <b>Note:</b> (see '\$G'). '***' modal groups are not changed (i.e. these values persist): Unit (mm or inch), Feed rate ('F'), Tool ('T'), spindle RPM ('S'). <i>Note: M2 does NOT restore Grbl to its startup modal values! The Linux-CNC standard is quite clear how M2 must function.</i>
M3	M3	Enable Spindle Clockwise	Enable spindle for clockwise rotation. Set Spindle RPM to 'S0' prior to calling M3, then call M3, then call desired RPM (e.g. S5000).
M4	M4	Enable Spindle CCW	M4 behavior is identical to M3, except spindle is enabled for counterclockwise rotation. Do not switch between M3 & M4 without first stopping spindle (i.e. setting RPM to 'S0').
M5	M5	Disable Spindle	Disable spindle rotation. Set spindle RPM to 'S0' prior to calling M5.
M17	M17 X-10	Stepper High Power Mode	Place all steppers into high power mode, starting at the next motion command, and continuing until the next time steppers become idle. USE SPARINGLY! PCB overheats (and safely shuts down) in ~5 seconds. Intended use: freeing bound X axis. Using M17 while the spindle is enabled could cause a brownout; M17 is a debug tool that consumes nearly half of Coast Runner's power budget.
M18	M18	Turn Off Steppers	Turn all steppers off until next motion command. Allows manual axis manipulation without unplugging Coast Runner. Steppers automatically re-enabled at beginning of next motion command.
M100	M100 G55 X G56 X G59 X	Find Midpoint & store in WCS	<i>(Supported in CRWrite Only) Syntax: M100 Gaa U Gbb V Gcc W</i> Find the midpoint between WCS Gaa's U axis and WCS Gbb's V axis, then store result in WCS Gcc's W axis. Example code finds the midpoint between G55's & G56's X axis values, then stores that midpoint into WCS G59's X axis value.
M101	M101 G55 G56 G59 X0.1	Verify probe results are within tolerance	<i>(Supported in CRWrite Only) Syntax: M101 Gaa Gbb Gcc Vnn</i> Verify the absolute difference between Gaa's V axis, Gbb's V axis, and Gcc's V axes are all less than nn. Alarm if calculated delta exceeds nn. The example code compares X values of G55, G56, and G59, and returns an error if any difference exceeds 0.1 units.
M102	M102 G54Y (G55Y+G56Y)/2	Perform arbitrary WCS math	<i>(Supported in CRWrite only) Syntax: M102 G5(4-9)(X,Y,Z) &lt;exprtk expression using variables of the form G5(4-9)(X,Y,Z)&gt;</i>  See notes below for details on M102.
M106	M106 G54X < 0 Error: G54X value is too low	Perform comparison and throw alarm on failure	<i>(Supported in CRWrite only) Syntax: M106 G5(4-9)(X,Y,Z) [comparison] [G5(4-9)(X,Y,Z), const] [Error message]</i>  See notes below for details on M106.
<i>Tn</i>	T1	Tool Number	Coast Runner doesn't have a tool changer. Used for tool offsets.

**Note:** Except for M17/M18/M100/M101/M102, all Coast Runner g-code commands conform to generally accepted industry syntax. When discrepancies exist between various g-code standards, Coast Runner attempts to follow LinuxCNC's g-code methodologies: <http://linuxcnc.org/documents/>

**Note:** This appendix is for reference only, and is not meant to be an exhaustive g-code resource. As with most things in life, g-code proficiency requires practice, patience, and experience. Entire textbooks exist on programming CNC machines; don't expect to master the art with just the above summary information.

**Note:** Grbl ignores text within parenthesis (i.e. comments). Keep comments to a minimum; grbl still has to read them into the serial buffer prior to discarding them. It's valid to send lines with only comments, or lines that are entirely empty.

**Note:** Grbl ignores white spaces (e.g. "G1 Y-50 F100 G91 G21" is identical to "G1Y-150F100G91G21")

### Notes on M102 (Arbitrary Math)

The M102 command is a special CRWrite-only command that allows you to perform arbitrary mathematical expressions within your G-code and to store the results in a WCS register. Other WCS registers can be used as input variables as well. This can be highly useful for doing things like finding the midpoint between two measured offsets, or adjusting an offset based on a previous measurement.

The command syntax is: *M102 G5(4-9)(X,Y,Z) <exprtk expression using variables of form G5(4-9)(X,Y,Z)>*

The first WCS register specifies the output register. Everything following the first register is treated as an expert expression. Exprtk is a C++ math library whose details can be found here: <http://www.partow.net/programming/exprtk/>. The expression passed in an M102 command must be well formatted and can take arbitrarily many input register variables that match the format described above.

Note that both the input registers and the output registers are *one* component of a WCS register - the X, the Y or the Z component. Only one component is operated on at a time. Valid registers are G54 - 59 and you can select the X, Y or Z component.

The expert expression is evaluated and the result of *expression.value()* is written to the output register. Nothing is written to the input registers during operation - only the specified output register is modified in an M102 command.

The M102 command is surprisingly powerful and should be used only by users who know what they are doing. Review the exprtk link to see the variety of supported features in the exprtk language.

### Notes on M106 (WCS Comparison)

The M106 command allows you to perform a comparison between a WCS register and either another WCS register, or with a constant. If that comparison fails, an error is thrown to display a custom error message. This can be very useful for checking assumptions or tolerances, or for any other use case requiring a comparison.

The command syntax is:

*M106 G5(4-9)(X,Y,Z) [comparison] [G5(4-9)(X,Y,Z), const] [Error message]*

This is rather complicated. Let's break it down:

- **M106:** the main Gcode command
- **G5(4-9)(X,Y,Z):** the first argument must always be a WCS register. You must select one or more components of the WCS register - X, Y, or Z - to be compared. *Multiple subcomponents may be selected*, meaning that G54X is valid, and so is G55YZ or G55XYZ. We will note more about selecting multiple subcomponents below.

- **[comparison]:** the comparison operator. This can be == (equals), <> (not equals), > (greater than), < (less than), >= (greater than or equal to), or <= (less than or equal to)
- **[G5(4-9)(X,Y,Z), const]:** the second argument can be either another WCS register (with one or more subcomponents) or a constant.
- **[Error message]:** The custom error message that will be displayed if the comparison fails.

#### **One to One Comparisons:**

In its most basic case, we can do a one to one comparison between a single WCS component and a constant. Here are valid one to one comparisons:

- *M106 G54X < 0 Error: G54X is too low*
- *M106 G55Y == 1 Error: G55Y should equal 1*

The first argument must always be a WCS value, never a constant. As such, this is **invalid**:

- *M106 0 > G54X Error: this is an invalid M106 command. Running this will throw a syntax error!*

#### **Like to Like Comparisons:**

You can also compare two WCS registers. We can do one or multiple comparisons at once, depending on the subcomponent. Here are some valid examples:

- *M106 G54X > G55X Error: G54X should be greater than G55X*
- *M106 G54YZ > G55YZ Error: G54 should have Y and Z components higher than G55*
- *M106 G54XYZ == G56XYZ Error: G54 and G56 should be identical*

#### **One to Many Comparisons:**

Finally, you can perform multiple comparisons where the subcomponents don't match. Here are some examples:

- *M106 G54XYZ >= G55X Error: at least one component of G54 is less than G55X*
- *M106 G55X < G54XY Error: either G54X or G54Y is less than G55X*

The M106 command was introduced in CRWrite 5.3.0 and is not supported in earlier versions.

## Appendix B: Grbl-Specific Commands

Coast Runner supports the following machine commands (Commands not typically used within prebuilt files are in gray):

Command	Function	Summary
\$H	Home Coast Runner	Find Coast Runner's origin using limit switches. All three axes are homed; Z is homed first, then X & Y are homed simultaneously. <b>Note:</b> Mill will not accept g-code until homed (unless unlocked).
\$HX, \$HY, \$HZ	Home Single Axis	Find Coast Runner's origin on specified axis only. If previously homed, both other axes' homing results are retained. <b>Note:</b> After homing any single axis, motion is allowed on all axes. <b>Note:</b> \$HX can crash the X Table if the Spindle is not retracted.
\$X	Unlock Coast Runner	Enable motion without homing. Strongly discouraged unless machine was previous homed this session (e.g. ok to unlock after sending Soft Reset). <b>Note:</b> Unlocking Grbl without previously homing arbitrarily sets the machine origin to the current position.
\$L	Auto-Level X table	See "12 - Leveling X Table"
\$LS	Store X Table as Level	DO NOT USE before reading "12.3 - \$LS Theory of Operation"
(shift+\) OR ctrl+x (0x18)	Soft Reset	Reset Grbl to power-up conditions, EXCEPT that previous machine position is retained. See "Recover After an Alarm" in Appendix E. <b>Note:</b> Character is a 'pipe' (shift+), not capital 'i' or lowercase 'L'.
\$I	Report Version Info	Show Grbl/PCB/VFD version information (see below).
\$E	Display EEPROM Map	Returns EEPROM map. Must connect to mill via 17.4.1 or 17.4.2.
?	Report Status	Show machine state, position, limit/probe status (see below).
\$G	Report Modal State	Show g-code parser state (see below).
\$C	Check G-code	Process g-code without motion. Grbl replies 'ok' if g-code line is valid. Useful to test g-code for syntax errors. \$C again to disable.
\$#	Report All Offsets	Show offsets (WCS, G28, G30, G92) & last probe result. WCS G54:G59, G28 & G30 are persistent, even across power cycles.
\$RST=#	Zero All Offsets	Zero all offsets (G54:G59 WCS, G28, G30, G92). Also resets all modal parameters (\$G) to startup values.
\$\$	Report Parameters	Show all stored parameters. See Parameters Table below.
\$RST=\$	Restore Parameters	Reset all parameters ('\$') back to OEM values.
\$RST=*	Restore All	Performs factory reset (by calling both \$RST=# and \$RST=\$).
\$Nk=___	Store Startup Line	Store a g-code line, which runs after each \$H. k=0,1
\$N	Display Startup Lines	Show stored started lines (see below).
\$n=___	Change Setting Value	Change a Grbl setting to a new value (see Appendix G)
!	Feed Hold	Axes in motion are decelerated to a stop. Spindle state is not affected. Linear motion not allowed until 'Resume' command sent. Not allowed while homing or leveling.
~	Resume	Resumes linear motion after a Feed Hold is sent.

**Note:** Most '\$' commands are primarily intended for use while developing new g-code.

### Grbl-Specific Commands - Response Syntax

Grbl's default startup response is:

```
Grbl 1.1h [help:'$']
```

Each time a serial command is sent during normal operation, Grbl performs the requested actions and then sends a response. Grbl replies to the vast majority of valid commands with a simple 'ok'. However, if the command syntax is invalid or illegal, Grbl responds with an error message (see Appendix E). If unexpected behavior occurs, Grbl will send an alarm message (see Appendix E) and then halt all operations until the interface is reset (see "Interfacing with Coast Runner").

The following commands send more information back to the host:

### Report Version Information - \$I

**Note:** Coast Runner returns a different \$I syntax than the Grbl standard (see Appendix H).

Send '\$I' to request Coast Runner's hardware/electronics/spindle versions, along with the firmware build date:

```
[Grbl:1.1h CR:3B PCB:3B VFD:3A YMD:20201212]
```

Example response:

```
[Grbl:1.1h CR:3B PCB:3B VFD:3A YMD:20201212]
```

Where:

'CR:' is the mechanical hardware version (e.g. 'CR:3B').

'PCB:' is the electrical hardware version (e.g. 'PCB:3B').

**Note:** These values are configured during the manufacturing and RMA process.

'Grbl:' is the base Grbl version that our firmware was forked from.

'YMD:' is the firmware build date.

**Note:** These values are configured during the firmware update process.

### Report Status - ?

**Note:** Coast Runner returns a different ? syntax than the Grbl standard (see Appendix H).

Send '?' to request Coast Runner's hardware status, with the following general syntax:

```
<state|Machine Coordinates (XYZ)|Buffer|Line Number|Probe X/Y/Z Limit Status|WCS Coordinates>
```

Element	Example	Description
State	Alarm, Idle, Run, Check	Machine State. Grbl starts in alarm state (until homed).
Machine Position	M:0.000,0.000,0.000	Absolute machine coordinates (based solely on limit switches). If unlocked (and not previously homed), initial location becomes 0,0,0.
Buffer	B:14,127	G-code buffer status (for debug). Shows remaining space.
Line Number	L:0	The current line number, if specified in g-code line (see Appendix A)
PXYZ	P0Y0	P: Probe is tripped. 0: Probe is NOT tripped. X/Y/Z: Limit switch is tripped. 0: Limit switch is NOT tripped.
WCS Offset	W:0.000,0.000,0.000	Displays the stored offset values for the current WCS. <b>Note:</b> This element is only sent every ten '?' requests.

Example Grbl responses:

```
<Alarm|M:-6.000,-0.500,-0.500|B:14,127|L:0|P0Y0> (probe is shorted and Y limit switch is tripped)
<Alarm|M:-6.000,-0.500,-0.500|B:14,127|L:0|0X00> (X limit switch is tripped, buffer is empty)
<Idle|M:-86.000,-0.500,-0.500|B:14,127|L:0|0000> (Machine is homed and ready for work)
<Idle|M:-86.000,-0.500,-0.500|B:14,127|L:0|0000|W:0.000,0.000,0.000> (10th '?' request; WCS sent)
```

**Note:** To determine limit switch status after a hard limit alarm occurs, send Soft Reset ('|'), then '?'.

### Report Parameters - \$\$

Displays Grbl's configured parameters. These parameters persist across power cycles. See Appendix G.

### Report Modal State - \$G

Reports the G-Code processor state. The default values are shown in the following example response:

```
[GC:G0 G54 G17 G21 G90 G94 M5 M9 T0 F0 S0]
```

Grbl supports the following modal groups:

Modal Group	Member Words
Motion Mode	<b>G0</b> , G1, G2, G3, G38.2, G38.3, G38.4, G38.5, G80
Coordinate System Select	<b>G54</b> , G55, G56, G57, G58, G59
Plane Select	<b>G17</b> , G18, G19
Units Mode	G20, <b>G21</b>
Distance Mode	<b>G90</b> , G91
Feed Rate Mode	G93, <b>G94</b>
Spindle State	M3, M4, <b>M5</b>
Coolant State	<b>M9</b> <b>Note:</b> Coast Runner doesn't use coolant; status always returns M9.
Tool Number	Tn Valid values are between <b>T0</b> & T255.
Feed Rate	Fn
Spindle RPM Setpoint	Sn Valid values are between <b>S0</b> & S8000. Larger values default to S8000.

### Report Stored Offsets - \$#

Reports each stored WCS offset (G54 through G59), G28/G30/G92 offset, tool length offset (TLO), and the last probe result that occurred during this session (PRB). Example response syntax:

```
[G54:0.000,0.000,0.000]
[G55:0.000,0.000,0.000]
[G56:0.000,0.000,0.000]
[G57:0.000,0.000,0.000]
[G58:0.000,0.000,0.000]
[G59:0.000,0.000,0.000]
[G28:0.000,0.000,0.000]
[G30:0.000,0.000,0.000]
[G92:0.000,0.000,0.000]
[TLO:0.000]
[PRB:0.000,0.000,0.000]
```

**Note:** G54/G55/G56/G57/G58/G59/G28/G30 persist until manually overwritten, even after power cycling.

**Note:** Send \$RST=# to zero these persistent values (see Appendix B for full details).

### Display EEPROM Map - \$E

Dumps Grbl's entire EEPROM space (0x0000 to 0x1008, x16 elements per line).

Explaining the EEPROM map is beyond the scope of this manual. Contact our support team for more information.

**Note:** This is a debug tool only. Do not send this command while Grbl is executing g-code.

**Note:** This command is only supported via a direct serial connection (see 17.4.1 & 17.4.2).

**Note:** Most Grbl-specific programs - including CRWrite - will hang if you send this command.

## Appendix C: Voltage Selector Switch

As shipped, Coast Runner accepts a standard 120 volt AC line input. If a standard 120 volt outlet is not available, then Coast Runner can be manually reconfigured to operate with a 240 volt AC line input. Follow the steps below to reconfigure Coast Runner to use 240 volt AC line input. There is zero advantage to running Coast Runner at 240 volts; Coast Runner has identical cutting power/torque in either configuration. See Appendix N page for full input voltage range.

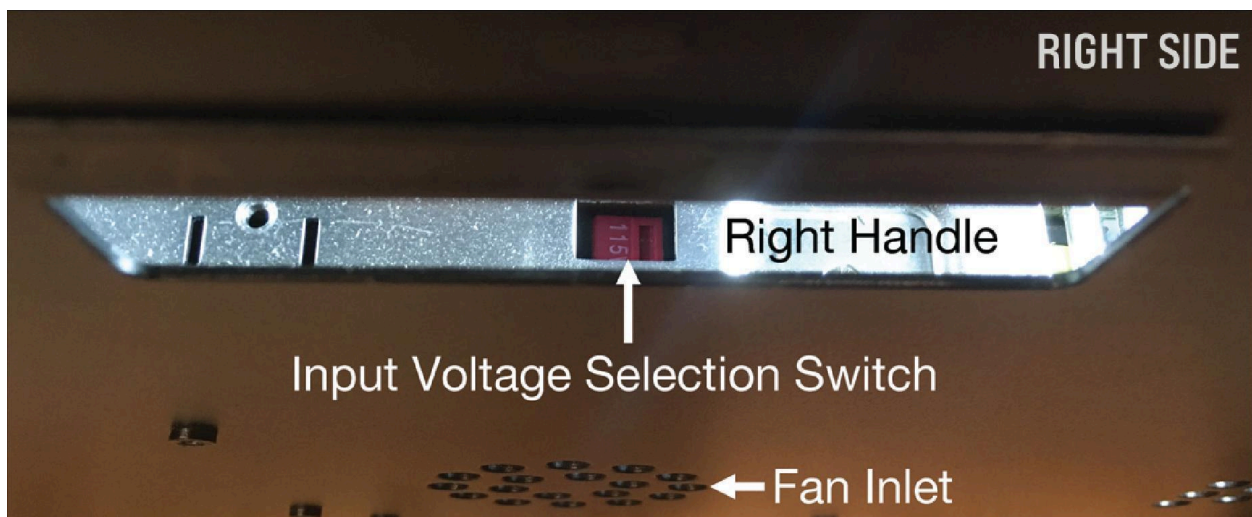
**Caution:** Failure to follow these steps when using a 240 volt AC line input could cause a fire. Coast Runner's built-in power supply requires manual reconfiguration - as outlined below - to operate safely at 240 V.

**Caution:** When a Coast Runner is sent in for service, our RMA team will revert the machine back to the 120 volt position. Therefore, when you receive the machine back, you will need to follow these steps again.

### Powering Coast Runner with a 240 volt AC line input

**Step 1:** Unplug the IEC power cable from the rear. Wait for the cooling fans to turn off.

**Step 2:** Locate the opening in the right handle. Shine a flashlight down into the opening. Identify the red switch.



**Step 3:** Insert a long flat head screwdriver down into the opening and slide the switch forward, until it reads '230V.'

**Note:** Some Coast Runner units have a thin plastic mesh covering the switch. The screwdriver should still be able to move the switch, but it might take more force. If unsuccessful, cut the mesh with a razor.

**Step 4:** Connect a NEMA 6-15P cable - with a C13 connector - into Coast Runner's mating C13 connector.

**Warning:** Do NOT cut, remove or bypass the safety ground prong. Do NOT plug the cable into an outlet that is not properly grounded. Defeating the safety ground terminal could result in electric shock (e.g. caused by stray chips shorting mains to the ungrounded metal enclosure). As if 120 volts wasn't enough, now you're playing the Darwin Award lottery with 240 volts. DON'T BE AN IDIOT!!!!!!!!!!!!

### Powering Coast Runner with a 120 volt AC line input

**Step 1:** Same as above.

**Step 2:** Same as above.

**Step 3:** Insert a long flat head screwdriver down into the handle and slide the switch rearward, until it reads '115V.'

**Step 4:** Connect the included power cable into Coast Runner's IEC connector.

**Warning:** Same as above... and yet some mastermind is still going to intentionally break the ground prong off their power cable, so that they can plug their Coast Runner into the non-grounded outlet in their mom's basement. Poor mom is going to come downstairs with tendies, only to find you dead.



# Appendix E: Errors & Alarms

## Errors

An error occurs if a g-code line contains one or more syntax errors. Grbl only reports the first encountered error. When an error occurs, Grbl performs the following steps:

- A: Discards the entire line (the syntax check is performed prior to executing any commands).
- B: Responds with three separate lines:
  - The first line echoes the entire command, with format "[echo:\_\_]" (e.g. "[echo: G0G1]").
  - The second line describes the error, with the format "[MSG:\_\_]" (e.g. "[MSG:G-code conflict]").
  - The third line is the specific error number, with the format "error:n" (e.g. "error:24").
- C: Executes next line in buffer. Errors do not place Grbl into an alarm state.

Error	Description
1	G-code words consist of a letter and a value. Letter was not found.
2	Missing the expected G-code word value or numeric value format is not valid.
3	Grbl '\$' system command was not recognized or supported.
4	Negative value received for an expected positive value.
5	Homing is not enabled via settings. Homing did not occur.
6	Minimum step pulse time must be greater than 3usec.
7	An EEPROM read failed. Auto-restoring affected EEPROM to default values.
8	Grbl '\$' command cannot be used unless Grbl is IDLE. Ensures smooth operation during a job.
9	G-code commands are locked out during alarm or jog state. Home or unlock required.
10	Soft limits cannot be enabled unless homing is also enabled.
11	Max characters per line exceeded (QTY79). Received command line was not executed.
12	Grbl '\$' setting value caused the step rate to exceed 30 kHz, which is not allowed.
14	Build info or startup line exceeded EEPROM line length limit. Line not stored.
15	Jog target exceeds machine travel. This manual does not discuss jogging.
16	Jog command invalid; has no '=' or contains unsupported g-code.
20	Unsupported or invalid g-code command found in block (selection conflict).
21	More than one g-code command from same modal group found (e.g. G0 & G1 in the same line).
22	Feed rate is not defined.
23	G-code command in block requires an integer value.
24	More than one g-code command that requires axis words found in g-code command.
25	Repeated g-code word found in line (e.g. G0 & G0).
26	No axis words found in g-code command that requires them.
27	Line number value is invalid.
28	G-code command is missing a required value, word, or number.
29	G59.x work coordinate systems are not supported.
30	G53 only allowed with G0 and G1 motion modes.

Error	Description
31	Unexpected axis word found in block when no command or current modal state requires them.
32	G2 and G3 arcs require at least one in-plane axis word.
33	Motion command target is invalid.
34	Arc radius value is invalid.
35	G2 and G3 arcs require at least one in-plane offset word.
36	Unused value words found in block.

## Alarms

An alarm occurs when an unexpected hardware condition exists.

When an alarm occurs, Grbl performs the following steps sequentially:

A: Immediately halts all motion. Any axis/axes in motion when the alarm occurs might lose steps, and thus the machine position (based on the last homing cycle) is only guaranteed on those axes that were not in motion when the alarm occurred. See "Position Guaranteed ?" in the table below.

B: Responds with three separate lines:

- The first line is a short human-readable description, with the format "[MSG:\_\_\_]" (e.g. "[MSG:limit Y]").
- The second line is a specific alarm number, with the format "ALARM:n" (e.g. "ALARM:1").
- The third line is a brief message "[MSG: Reset to cont]".

C: Enters the alarm state, and will not process, buffer, or respond to any command, except for the Soft Reset command ('|', see Appendix B). See "Recover After an Alarm" below.

Alarm	Description	Position Guaranteed?
1	Hard limit: An axis ran into a limit switch.	No
2	Soft limit: Requested motion would exceed machine travel.	Yes
3	Reset occurred while in motion.	No
4	Probe is not in the expected starting state.	Yes
5	Probe did not trip within the specified travel distance.	Yes
6	Reset occurred during homing routine (e.g. Soft Reset sent)	No
8	Limit switch did not reset during homing routine.	No
9	Limit switch did not trip during homing routine.	No

## Recover After an Alarm

When running known-working g-code, an alarm indicates something has gone seriously wrong. If CRWrite reports an alarm while running a known-good file, then the "recovery method" is to start over.

When developing new cutting code, g-code mistakes are inevitable. For example, if the machine units are set to inches, and you send Y-40 (thinking you're in mm), then a soft limit alarm will occur (40 inches exceeds Coast Runner's maximum Y travel). Since an Alarm prevents grbl from processing further commands, you can quickly recover from this alarm by sending the Soft Reset pipe command ('|').

Sending the Soft Reset command causes Grbl to reinitialize - as if Coast Runner were just plugged in - EXCEPT that the machine position is retained (from the previous homing cycle). You can then send '\$X' to unlock the machine, retaining the previous machine zero (without homing), or you can '\$H' to re-home the machine (if desired).

**Note:** Machine position is the only volatile information retained after a Soft Reset occurs.

**Note:** Non-volatile persistent information (e.g. WCS, machine settings, etc) is also retained.

## Appendix G: Grbl Settings (control parameters)

Setting	Function	Valid Values	Unit	Summary
\$1	Stepper idle delay	100, 255	ms	100 allows steppers to enter low power mode when stationary. Homing position is maintained. 255 prevents low power stepper mode.
\$13	Report inches?	0,1	mm,in	0 causes mill to report all coordinates in mm 1 causes mill to report all coordinates in inches <b>Note:</b> All internal math is still performed in mm; Grbl simply reports all results in inches if \$13=1. <b>Note:</b> We recommend leaving this setting alone. To use inches in G-code, use G20 instead.
\$20	Enable soft limits?	0,1	-	<p>Prevent motion commands from executing that would move an axis beyond its mechanical travel limits, based on the last homing cycle (\$H). Useful since each axis only has a limit switch on one side.</p> <p>When enabled, all motion commands are compared to the maximum gantry travel parameters (see \$130, \$131, \$132). Specifically, all motion requests must reside between 0 and the negative value stored for each axis (e.g. X values must be between -86.5 &amp; 0 mm). If a given motion command would exceed mill's mechanical limits (i.e. crash into the unmonitored side), then no motion occurs and an alarm is generated.</p> <p><b>Note:</b> See "Appendix E: Recover After an Alarm" for steps to retain previous position without rehomings.</p> <p><b>Note:</b> Soft limits only prevent crashing if all three axes were previously homed (\$H), AND no steps have been lost (e.g. crash, steppers powered off). If mill is unlocked (\$X) - without having homed during the same session - each axis' current position is set to zero (i.e. the current machine position becomes &lt;0,0,0&gt;). Under these conditions, if soft limits are enabled and:</p> <ul style="list-style-type: none"> <li>-a <u>positive</u> machine position command is sent, then a soft limit error will occur because soft limits require all motion to occur in negative space.</li> <li>-a <u>negative</u> machine position command is sent, then a crash will occur if the specified move is BOTH greater than the actual remaining negative axis travel AND less than the maximum gantry travel parameters (e.g. \$130).</li> </ul> <p>With soft limits disabled, mill does not check g-code positions and will happily ram into the non-switched side, thus losing position.</p> <p><b>Note:</b> If hard limits are enabled, tripping any axis limit switch always immediately stops all linear motion.</p>

Setting	Function	Valid Values	Unit	Summary
\$21	Enable hard limits?	0,1	-	If enabled, tripping a limit switch will immediately cease all motion (except while homing)
\$22	Require homing prior to accepting g-code?	0,1	-	If enabled, mill must be homed (\$H) or unlocked (\$X) prior to executing g-code. There's almost no reason to disable (\$22=0)... send \$X instead.
\$30	Max spindle RPM	8000	rev/min	Changing this value won't magically increase spindle RPM. Recommendation: Don't modify
\$110 \$111 \$112	Max X velocity Max Y velocity Max Z velocity	100 to 4500	mm/min	Maximum velocity allowed on each axis. Some units can run at higher velocities. Default values are conservative, to ensure mill won't lose steps during normal operation. Increase at your own risk.
\$120 \$121 \$122	Max X acceleration Max Y acceleration Max Z acceleration	10 to 1000	mm/min <sup>2</sup>	Maximum acceleration allowed on each axis. Some units can run at increased acceleration, which allows an axis to reach maximum velocity over a shorter distance. The default values are conservative (to ensure mill won't lose steps during normal operation). Increase at your own risk.
\$132	Max Z travel	78.5	mm (ONLY)	When soft limits are enabled, defines maximum travel allowed on each axis. <i>Tip: Don't modify.</i>  <i>The Z limit switch is physically located in the <u>positive</u> direction of travel. When the Z limit switch trips during the homing routine, that point becomes Z=0 (because mill operates in negative space).</i>
\$131	Max Y travel	241.5	mm (ONLY)	<i>The Y limit switch behavior is identical to Z.</i>
\$130	Max X travel	86.5	mm (ONLY)	<i>The X limit switch is physically located in the <u>negative</u> direction of travel. When the X limit switch trips during the homing routine, that point does NOT become X=0, but instead is set to X=-[\$130] (e.g. X=-86.5). Therefore, be particularly careful when modifying Max X travel, because that will change where the X Table physically positions itself given the same absolute position command.</i>  <i>By default, sending "G53 X0" moves the X table all the way down. However, if Max X travel is changed to 43.25 mm (i.e. \$130=43.25), then sending G53 X0 will move the X table directly to the middle of travel (because X=0 is +43.25 mm from where the X limit switch tripped).</i>

**Note:** To change a setting, type '\$n=y', where 'n' is the setting to change, and 'y' is the new value.

**Note:** Changes to these parameters are persistent, even across power cycle (until overwritten).

**Note:** Many additional Grbl parameters are not shown because they should not be modified. For a complete explanation of all parameters, see: [github.com/gnea/Grbl/wiki/Grbl-v1.1-Configuration](https://github.com/gnea/Grbl/wiki/Grbl-v1.1-Configuration)

**Note:** Performing any of the following actions will restore all parameters to default values:

- Pressing "Restore Machine Defaults" in CRWrite (Settings>Machine Actions)
- Sending '\$RST=\$' or '\$RST=\*'
- Updating Coast Runner firmware (but ONLY if existing settings are invalid).

## Appendix H: Differences Between Grbl & Our Fork

Coast Runner runs a fork of the original grill ([github.com/gnea/grbl](https://github.com/gnea/grbl)). From an interfacing perspective, our fork is identical to Grbl, except as noted below:

Condition	Grbl Behavior	Fork Behavior	Interface Changes	For Further Information
Stepper Motor Power Levels	Either on or off. (on generates excess heat) (off can lose steps)	Configurable high/norm/low/off (level automatically selected) (manual control available)	None	M17 M18
"ALARM:" Messages	Code only (e.g. "ALARM:1") (no human readable text)	Code (e.g. "ALARM:1") & Human readable text (e.g. "[MSG:LIM X]")	None	Appendix E
Alarm Behavior	Auto-reset (except limits)	Requires manual Soft Reset	Send Soft Reset after alarm	
Soft Reset	"[control+x]"	"[control+x]" OR " " (i.e. "[shift+ ]", aka a 'pipe')	None	Appendix E: "Recover After an Alarm"
"error:" Messages	Code only (e.g. "error:9") (no human readable text)	Code (e.g. "error:9") & Human readable text (e.g. "[MSG:\$H]")	None	Appendix E
Response to invalid g-code Syntax	"error:2"	"[echo:_____]" (echoes input) "[MSG:_____]" (guess at issue) "error:2"	None	Appendix E
Auto-Level \$L	n/a	Automatically squares the X axis	None	"Leveling X Table"
Probing	polling (trip less reliable)	interrupt-driven (trip guaranteed)	None	G38
Coolant	M7/M8/M9 supported	M7/M8/M9 allowed, but Coast Runner lacks coolant so no action occurs	None	-
\$I Formatting	Grbl version + options	Grbl version + Mill info	None	Appendix B
'?' Formatting	See Grbl documentation*	Probe & Limit status is either '0' (not tripped) or 'P', 'X', 'Y', 'Z'	Parsing Syntax	Appendix B

Except for the differences listed above, the fork's interface is identical to Grbl's documentation:

\*<https://github.com/gnea/Grbl/wiki/Grbl-v1.1-Interface>

## Appendix K: Disassembling Coast Runner

Gaining access to Coast Runner's internals isn't required during normal use. If you're reading this section, you're either:

- A) having technical problems (sorry!), or
- B) curious about Coast Runner's internals (cool!), or
- C) just one of those people that likes to read manuals (sweet!).

Whatever the reason, the steps below outline how to remove the outer sheet metal enclosure.

**Note:** Failure to follow these disassembly steps exactly as shown could damage Coast Runner's internal components. Proceed at your own risk... with great power comes great responsibility!

### Tools required:

- Allen key - 3.0 mm - square end
- Allen key - 2.5 mm - ball end
- Allen key - 2.5 mm - square end - 8" minimum shaft length (12" recommended). Recommendations:  
BTI B00178H1D8 (best); Bondhus 25445 (good); EKLIND 54925 (ok).

**Note:** All M3 bolts removed during disassembly are identical.

**Note:** All M4 bolts removed during disassembly are identical.

**Step 1:** Unplug the power and USB cables, vacuum all metal chips, then place Coast Runner upside down on a table.

**Note:** Do NOT use compressed air to remove chips from Coast Runner.

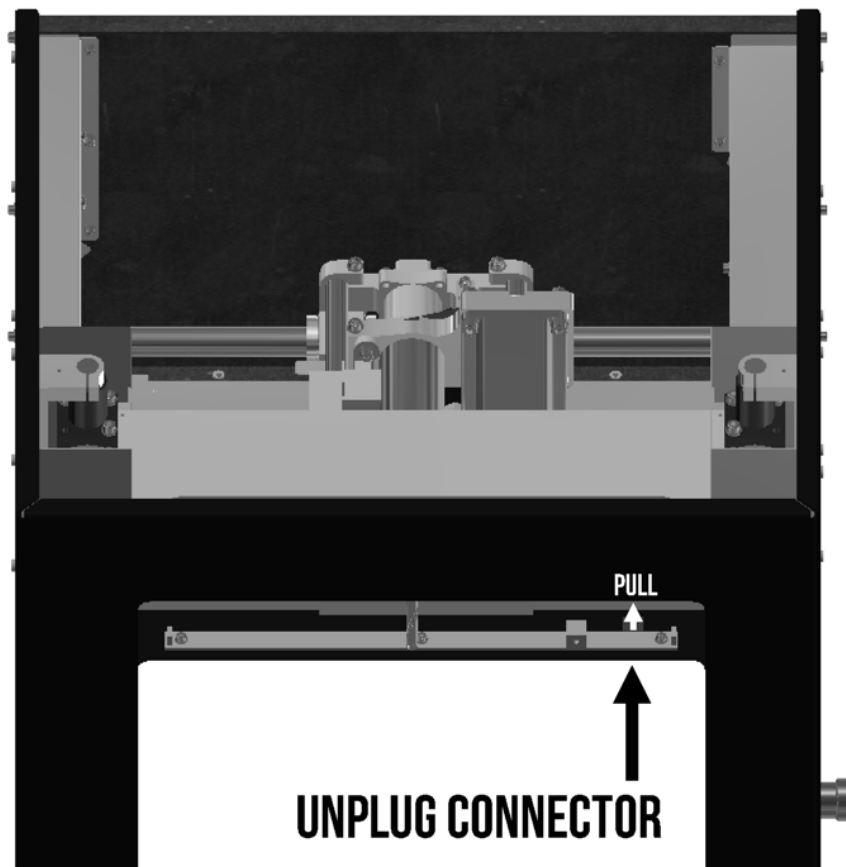
**Note:** Do Not invert Coast Runner prior to vacuuming (chip management is gravity based).

**Step 2:** Unplug LED PCB connector (by pulling in the Z+ direction).

**Note:** Take a picture of the wiring before unplugging the connector.

**Note:** Moving the X table away from the spindle makes it easier to unplug the cable.

**Note:** Moving the Y gantry to the left makes it easier to unplug the cable.

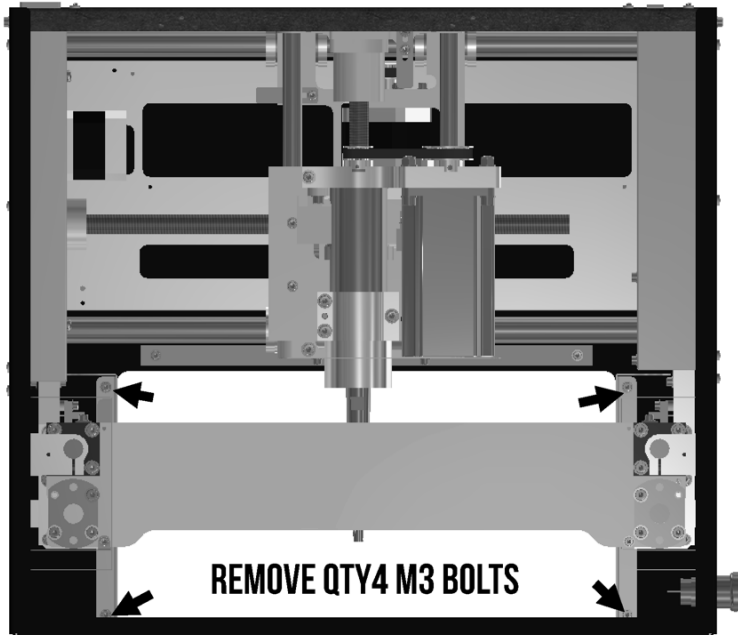


**Note:** This is a friction connector (i.e. it does not have retaining tabs; just pull).

**Step 3:** Remove QTY4 M3 Bolts, as shown below. Use a long 2.5 mm square Allen key.

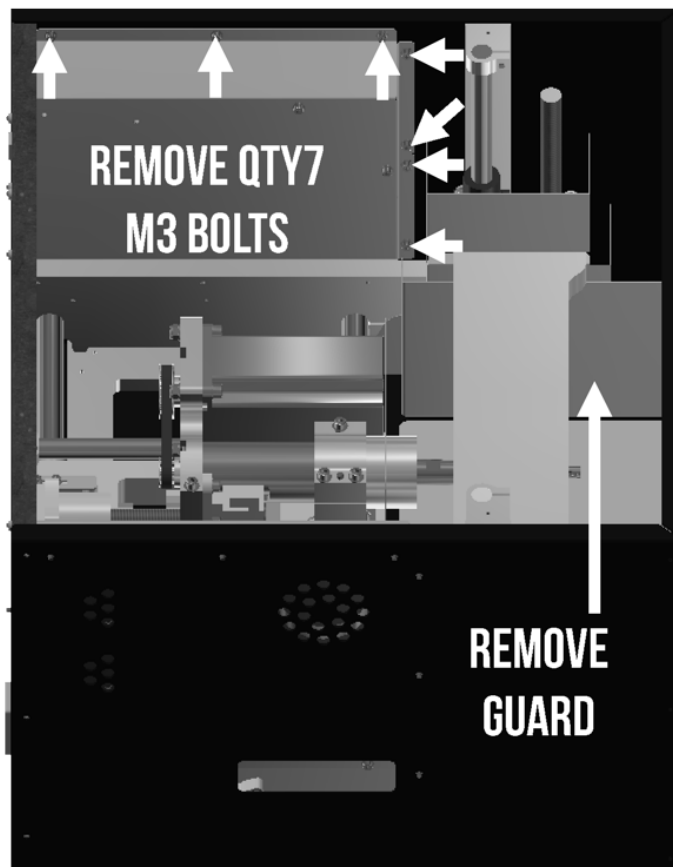
**Note:** Pushing the X table towards the spindle reduces the required Allen key length.

**Note:** Using a shorter Allen key requires bending the aluminum chip guards (not recommended).



**Step 4a:** Remove QTY2 aluminum chip guards (bolts removed in previous step).

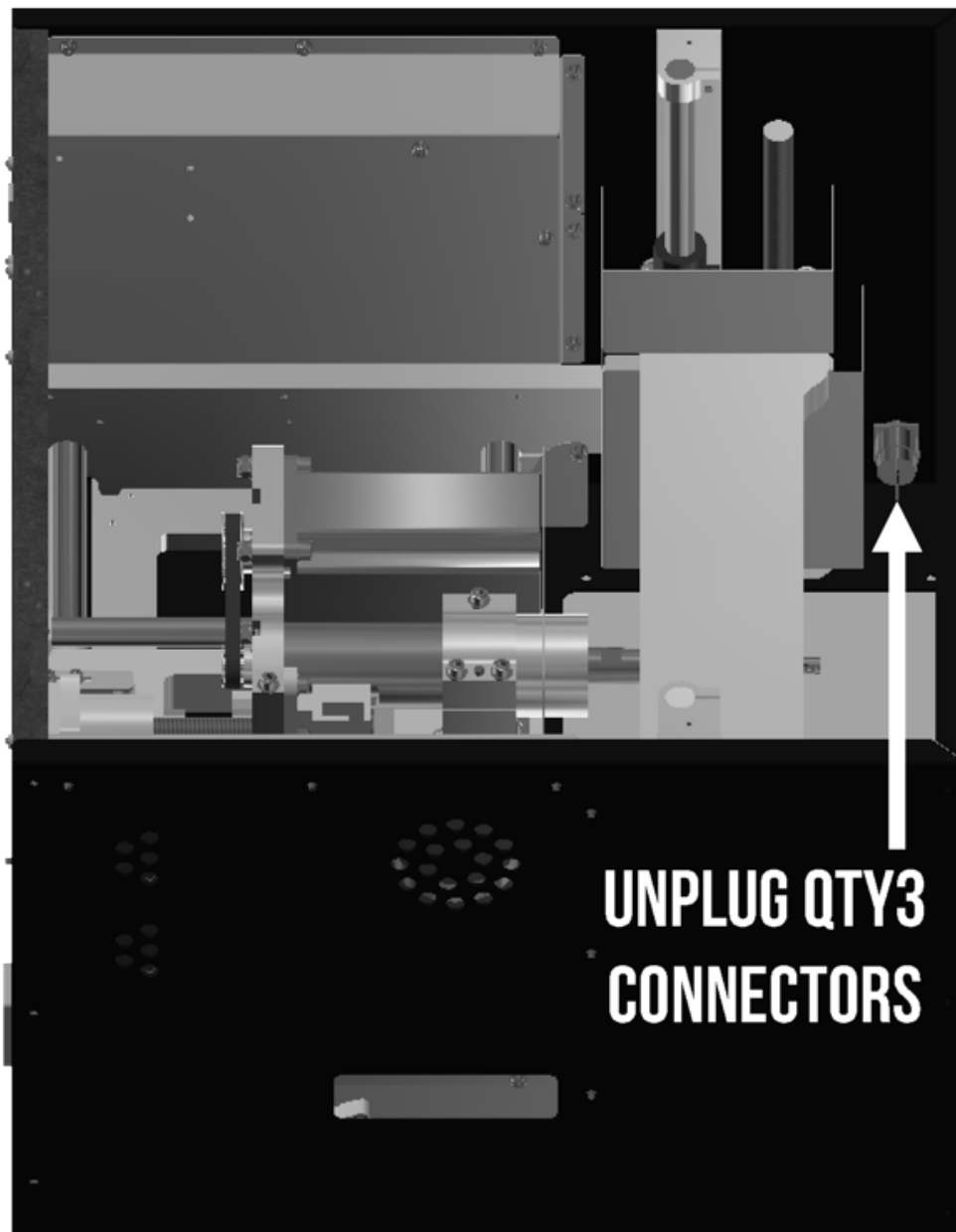
**Step 4b:** Remove QTY7 M3 bolts from electronics bay cover.



**Step 5:** Remove QTY3 spade connectors from emergency stop switch.

**Note:** Take a picture of the wiring before removing the connectors!

**Note:** Verify the USB cable is unplugged prior to removing these connections.



**Note:** Communication with Coast Runner is impossible when the E-stop leads are disconnected. If axis/spindle motion is required to troubleshoot an issue while the cover is removed, temporarily connect 'NC' to 'C'

(see step 5 of the reassembly instructions for information on how to determine which wires to connect).

**Warning:** When the E-stop switch is bypassed, Coast Runner can theoretically move whenever both the power & USB cables are connected. Stay clear of the axes, spindle, and all belts whenever power is applied!



**Step 6:** Remove QTY6 M3 bolts from power supply cover.

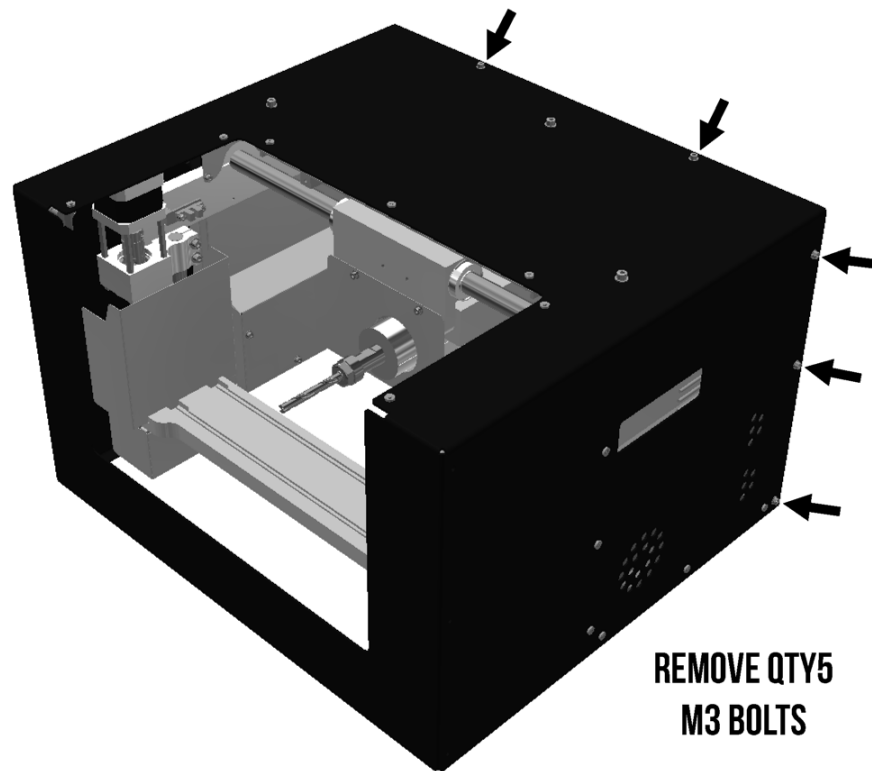


**Danger:** Hazardous voltages present when this cover is removed! The brown and blue wires routed from the IEC connector housing to the power supply behind this cover are directly connected to high voltage whenever the external power cable is energized. Do not contact any exposed portion of these wires, including the point where they interface to the large power supply.

**Danger:** Do not energize the IEC power connector if the green/yellow wire is disconnected from the power supply. This is the safety ground conductor path that will prevent Coast Runner from electrocuting you if the enclosure becomes energized. Remove power from Coast Runner before servicing the power supply!

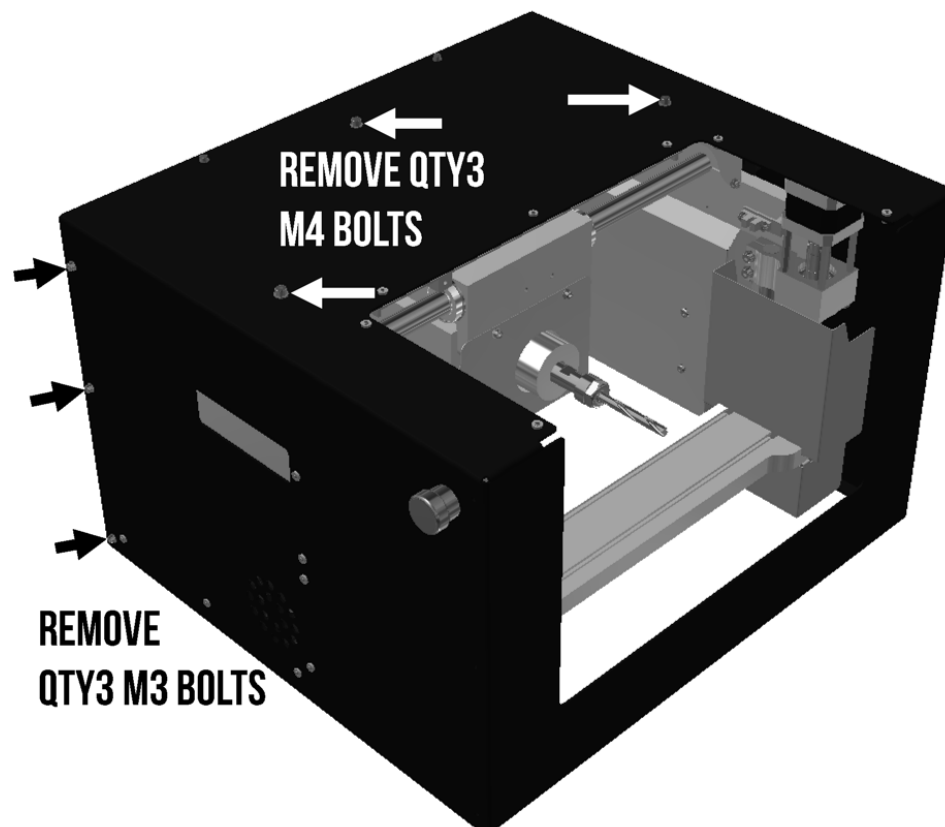
**Note:** All other wiring inside Coast Runner is less than 30 volts and is safe to touch with bare hands.

**Step 7:** Remove QTY5 M3 bolts

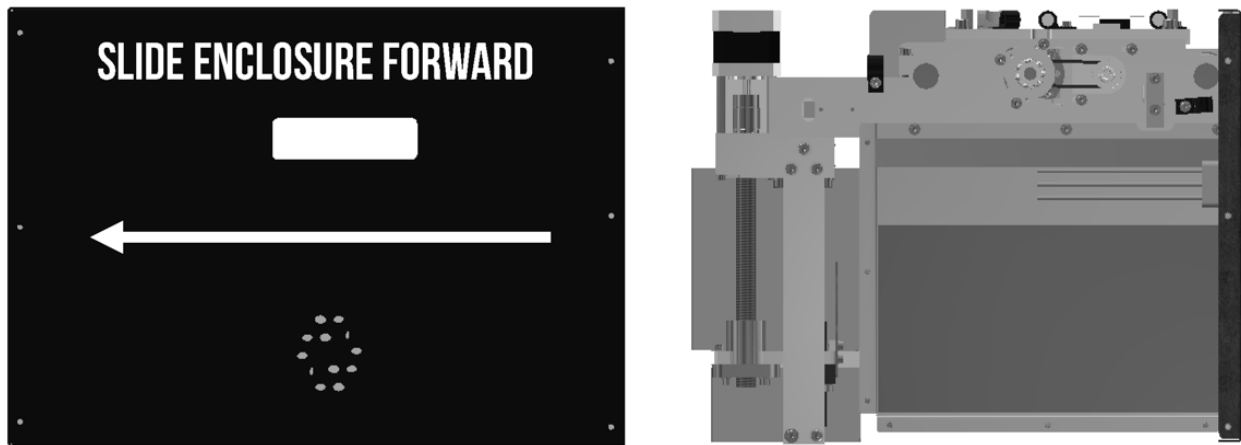


**Step 8a:** Remove QTY3 M3 bolts, then;

**Step 8b:** Remove QTY3 M4 bolts



**Step 9:** Remove outer enclosure by sliding forward until it is free from Coast Runner internals.



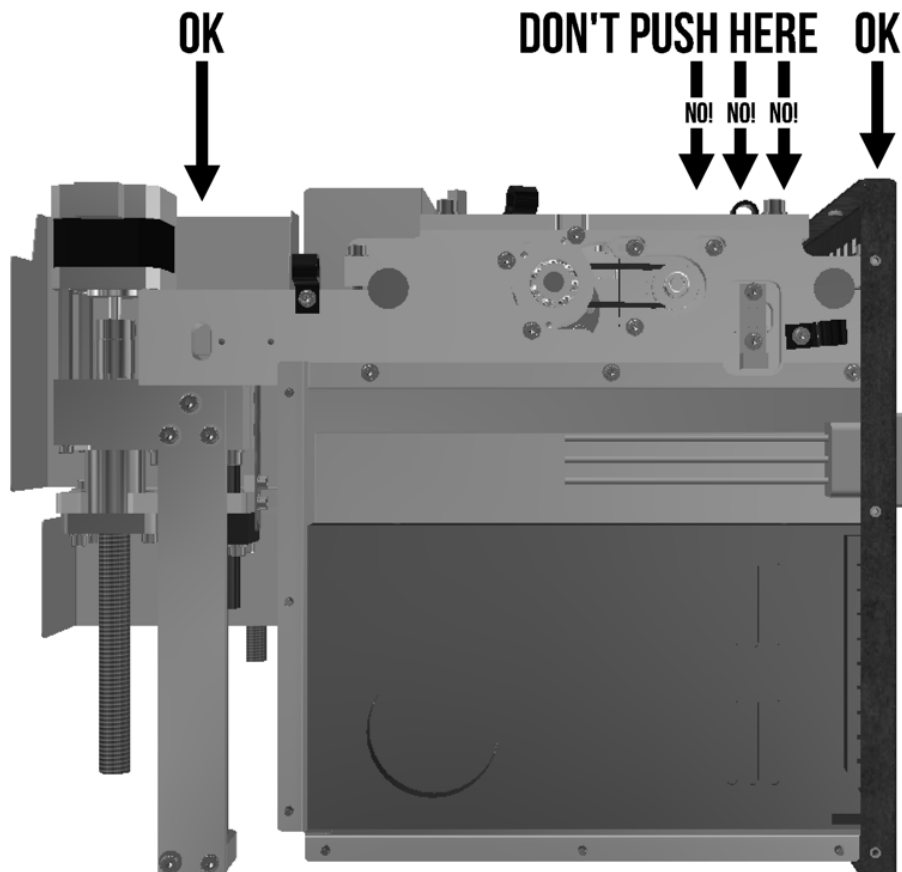
With the outer enclosure removed, Coast Runner's internals are supported by:

- Front: The aluminum X arms, which are designed to handle force.
- Rear: The rear, sheet steel cover, which attaches to the internal structure via two thin aluminum covers.

**Note:** The thin aluminum covers are NOT designed to support external forces.

Therefore, when the outer enclosure is removed:

- DO NOT PRESS DOWN ON THE REAR OF THE MACHINE !
- DO NOT PICK UP COAST RUNNER BY THE REAR COVER !



## Reassembling Coast Runner

Before reassembling, spend a few minutes vacuuming stray chips from the machine. During normal operation, the chip guards are designed to prevent chips from entering the machine. However, during the disassembly process, some chips might find their way past these seals and into the sensitive electronic/mechanical components.

**Note:** DO NOT USE COMPRESSED AIR; VACUUM ONLY!

To reassemble Coast Runner, follow the disassembly steps in reverse order, with the following additional steps:

**Note:** These steps are intentionally in reverse order, to correspond with disassembly steps.

**Step 9:** Before sliding the enclosure on, tape the emergency stop cable to the left X Table's aluminum chip guard.

**Note:** If you don't do this now, you'll end up fishing them out later.

**Step 8:** Leave all bolts loose until all bolts are installed (i.e. leave loose until step 4).

**Note:** To align holes, place a 1/4" object under the rear cover, then apply downward force on enclosure.

**Step 7:** Leave all bolts loose until all bolts are threaded (i.e. leave loose until step 4).

**Step 6:** Leave all bolts loose until all bolts are threaded (i.e. leave loose until step 4).

**Step 5:** Be sure to connect each spade terminal to the correct E-stop terminal.

If unsure, verify all three wires are separated and not touching any metal surface, then plug in the USB cable (only) and measure the voltage between the X Table and each disconnected wire. The correct pinout is:

Wire with ~0.0 volts connects to 'NO' terminal.

Wire with ~4.7 volts connects to 'NC' terminal.

Wire with ~0.7 volts connects to 'C' terminal.

**Note:** Terminal labels are molded into the E-stop switch, near each medal connector.

After all three connectors are installed, verify Coast Runner can communicate with CRWrite:

A: Leave the IEC power cable unplugged throughout this test.

B: Plug a USB cable into Coast Runner.

C: Engage the emergency stop.

D: Open CRWrite. CRWrite should report that the reset button is engaged.

E: Disengage the emergency stop button by twisting clockwise.

F: CRWrite should now connect to Coast Runner.

If the above behavior is not observed:

-Hold the negative lead of a voltage meter to the X Table.

-Hold the positive lead onto the 'C' terminal.

-When the E-stop switch is 'up' (not engaged), 'C' should measure  $4.7 \pm 0.5$  volts.

-When the E-stop switch is 'down' (engaged), 'C' should measure  $0.5 \pm 0.5$  volts.

**Step 4:** Leave all bolts loose until all bolts are threaded, then tighten all bolts (including those in steps 8,7,6).

**Step 3:** Installing these bolts is easier if the long Allen key is magnetized (or dipped in heavy grease).

**Step 2:** Be sure to plug the connector in exactly as it was installed before. To verify proper installation:

A: Plug a USB cable into Coast Runner.

**Note:** Do not plug in the IEC power cable until this connector is properly installed!

B: Plug the Probe Cable into the Probe Connector.

C: Touch the Probe Cable's ring terminal to the X Table. The Probe LED must illuminate.

**Note:** The probe WILL NOT WORK if the LED doesn't illuminate.

D: Remove the Probe Connector's ring terminal from X Table. The Probe LED must turn off.

If the above behavior is not observed:

-Verify the 5-pin connector is plugged in properly (try flipping it the other way).

**Note:** As long as the IEC power cable is not plugged in, this connector can be incorrectly plugged in without risking damage to Coast Runner's electronics. Plugging in the IEC power cable while this connector is incorrectly installed could damage Coast Runner's electronics.

**Step 1:** If you manually moved the X Table at any point during this process, then the X Table might not be level. To automatically re-level, send the command '\$L' to Coast Runner. See "Leveling X Table" for more information.

## Appendix L: Replacing Spindle Belt

The spindle belt is a consumable item, although we haven't seen one fail outside brutal testing:

- We operated the spindle at full load for over eight hours (using a purpose-built dynamometer). This is equivalent to mechanically transferring 3 MJ through the belt before it shredded.
- We drilled QTY400 0.375" diameter holes through 3" solid red oak (100 feet in total), then peck drilled QTY100 0.219" holes through 1/4" stainless steel (25 inches in total).

Coast Runner uses Gates 172-2mgt-06 belts, which are far superior to most GT2 belts. Some GT2 belts we tested didn't last even 5% as long as the Gates belt. Contact our support team to obtain a replacement belt.

To replace the spindle belt:

A: Home Coast Runner, then center the Y, drop X down, and then plunge Z:

>\$H

>G53 G90 G21 X0Y-120

>G53 Z-78.5

B: Unplug the USB and power cable.

C: Remove all chips from Coast Runner with a vacuum.

D: Rotate Coast Runner upside down, with the front opening facing you.

E: Identify the BLDC motor (just right of the spindle).

F: Loosen QTY4 M5 bolts with a 4 mm Allen and an 8 mm wrench.

**Note:** Don't remove the bolts, just make them slide freely.

**Note:** To access the bolt head between the motor and spindle, slide the long end of a 4 mm Allen into the small opening visible when looking into Coast Runner's opening from the side. Navigate the tool tip above the optical limit switch (white rectangular connector). Once the Allen tip is mated to the bolt head, rotate the nut with the 8 mm wrench to loosen the bolt.

G: Remove the old belt.

H: Place new belt around sheaves. If the belt is too tight, loosen the bolts more... eventually the belt will fit.

J: Snug the bolts until the BLDC motor is able to barely slide left/right.

K: Tension the belt by pulling the BLDC motor to the right.

**Note:** We apply tension by placing a 0.250x1x6" aluminum bar between the spindle body and BLDC.

L: While applying tension, tighten the top bolt closest to the spindle, then tighten the other top bolt.

M: While maintaining tension, tighten the easily accessible bottom bolt.

N: Remove tension, then tighten the hidden bolt.

P: Verify the belt is properly tensioned. It should make an audible 'ping' when plucked.

## Appendix N: Specifications

Parameter	Value	Unit	Note
X/Y/Z Travel Distance	86.5/241.5/78.5	mm	~3.4/9.5/3.1 inches
Total cutting volume	1636	cm <sup>3</sup>	~100 cubic inches
X/Y/Z Working Space	116/299/162	mm	With Chip Cover installed
Total working volume	5618	cm <sup>3</sup>	~343 cubic inches
Max Linear Velocity: X Max Linear Velocity: YZ	2540 3100	mm/min mm/min	~100 ipm ~122 ipm
Max acceleration	500	mm/s <sup>2</sup>	~20 in/s <sup>2</sup>
Weight	19.5	kg	~43 pounds
Operating Temperature	0 to 40	C	32 to 104 F
Storage Temperature	-20 to 55	C	-4 to 131 F
Relative Humidity	0-50	%	0-80% with lubrication (see "11 - Maintenance")
USB Current	250	mA	Requires Powered USB port/hub
Input Voltage (OEM)	90-132	Vac	As shipped
Input Voltage (available)	180-264	Vac	<b>Warning:</b> See Appendix C
Input Power	500	W	maximum power consumed by Coast Runner
Input Frequency	47-63	Hz	Does not affect spindle RPM
Spindle runout	0.0009	in	Measured on tapered collet, 1 mm from tip
X Table Clamping Method	2	T-Slots	For use with 16 mm M4-threaded T-Nuts
T-slot spacing	40	mm	center-to-center
Collet system	ER11	-	Tool diameters from 1 to 8mm (0.040 to 0.315")
Max Tool length	82.5 (3.25)	mm (in)	Longer tools will crash while homing
Spindle Input Power (max)	375	W	Electronically limited by VFD
Spindle Input Power (rms)	190	W	Long-term average to prevent overtemp
Spindle Cutting Power (max)	225	W	The peak cutting energy available for the tool to remove material. Exceeding this value will cause VFD to limit power automatically.
Spindle Cutting Power (rms)	100	W	The <b>continuous</b> cutting energy available for the tool to remove material, without overheating.
Spindle RPM (guaranteed) Spindle RPM (typical)	3000-8000 1000-9000	RPM RPM	-
Supported Operating Systems	10.11 (or later) 7 SP1 (or later)	Mac Win	We test in 10.13, 10.14, 10.15 We test in 7x64, 8x64, 10x64

## Appendix P: Axis Labels

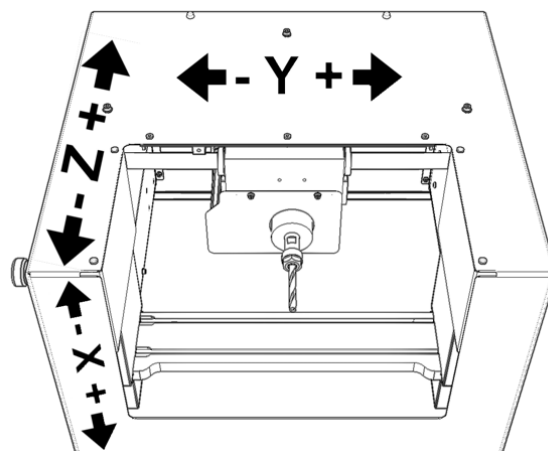
If desired, cut out these labels and tape them to Coast Runner as shown to the right. As with most CNC machines, Coast Runner operates in negative coordinate space:

Y spans from 0\* mm to -241.5 mm

Z spans from 0\* mm to -78.5 mm

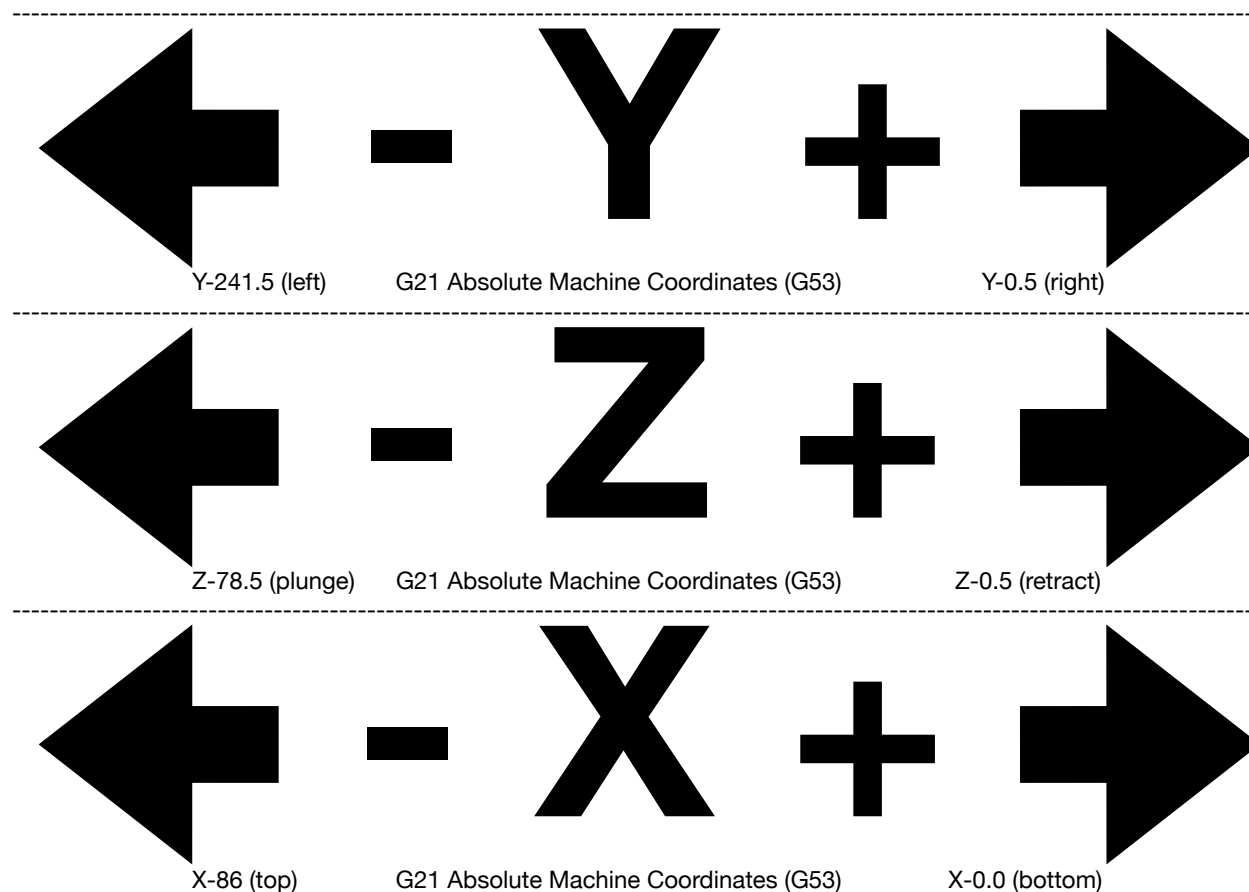
X spans from 0 mm to -86.5\* mm

**Note:** Moving exactly to the '\*' values (above) could result in a hard limit error. During the homing process, the point where the X/Y/Z limit switch trips becomes exactly -86.5/0/0 in coordinate space, respectively. To prevent unexpected hard limit errors, stay 0.5 mm away from each limit switch. The numbers below reflect this guard band, and are the recommended travel range.



The values listed in this appendix assume the following modal settings:

- Unit mode is millimeters (G21). **Note:** Grbl defaults to this value at the beginning of each session.
- Motion mode is absolute (G90). **Note:** Grbl defaults to this value at the beginning of each session.
- WCS mode has no offset from absolute machine coordinates. There are two ways to achieve this:
  - If you've followed our advice to not store offsets in WCS G54, then use WCS mode G54.
  - Note:** If you haven't followed this advice, see Appendix B: \$RST=#.
- Add G53 to each manual g-code motion line. G53 forces Grbl to move in absolute machine coordinates.
- Note:** G53 only applies to the line containing it (see Appendix A).



## Physical X Table Movement