



Speeduino Miata / MX5 Manual

This manual is compiled from the Speeduino documentation wiki:

<https://speeduino.com/wiki/index.php>

Sat Feb 16 2019

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Connecting to TunerStudio	1
1.2.1	Downloading Tuner Studio	2
1.2.2	Setting up your project	2
1.3	Creating a TunerStudio Project	5
1.3.1	Configuring TunerStudio Project Properties	5
1.3.2	Settings Tab	6
1.3.3	Can Devices Tab	8
2	Hardware	9
2.1	MX5 PNP	9
2.1.1	Introduction	9
2.1.2	Hardware requirements	9
2.1.3	Configuration and start	13
3	Configuration	15
3.1	Engine Constants	15
3.1.1	Overview[edit]	15
3.1.2	Settings[edit]	16
3.2	Injector Characteristics	17
3.2.1	Overview	17
3.2.2	Settings	17
3.3	IAT Density	17
3.3.1	Overview	17
3.3.2	Settings	18
3.4	Acceleration Enrichment	18
3.4.1	Overview	18
3.4.2	Theory	18
3.5	Rev Limits	19
3.5.1	Overview	19
3.5.2	Settings	20
3.6	Flex Fuel	20

3.6.1	Overview	20
3.6.2	Hardware	20
3.6.3	Tuning	21
3.7	Spark Settings	22
3.7.1	Overview	22
3.7.2	Settings	22
3.8	Dwell	24
3.9	Overview	24
3.10	Settings	25
3.10.1	Voltage correction	26
3.11	Cranking	26
3.11.1	Overview	26
3.11.2	Settings	27
3.12	Warmup	28
3.12.1	Overview	28
3.12.2	Settings	28
3.13	Idle	29
3.13.1	Overview	29
4	Updating firmware	36
4.1	Compiling and Installing Firmware	36
4.1.1	Latest Stable Firmware	36
4.1.2	Installation - Easy Method	36
4.1.3	Manually Compiling	36

Chapter 1

Introduction

1.1 Introduction

This manual covers the hardware (sensors, wiring etc), software configuration and tuning elements related to running a Speeduino unit. When beginning with Speeduino, particularly if it is your first time installing and configuring an engine management system, this manual will assist in understanding Speeduino's capabilities and how it should be installed, both in terms of hardware and software/firmware.

Whilst this document will assist in providing information related to Speeduino's configuration, it does not cover advanced engine tuning, fuel / ignition strategies etc. As with any changes to engine management, the possibility of damage to hardware is very real should a system be configured incorrectly.

Getting Started

In terms of starting out with Speeduino, it is generally recommended to first upload the firmware to your Arduino and get it connecting to the tuning software (Tuner Studio) before moving on to hardware assembly or wiring etc. Software setup and configuration on Speeduino can be completed without the need for any additional hardware to be present (Beyond the arduino itself) and this allows exploration of the software and options available before either an outlay of significant funds or a significant investment of time.

The following sections of this manual cover how to compile and upload the firmware, as well as creating a new project in Tuner Studio. It is strongly recommended to read these as a starting point.

About this manual

The contents of this manual are compiled from the Speeduino wiki at <https://speeduino.com/wiki/index.php>. As an open source project, this documentation is growing continually and this offline manual is updated nightly with any changes that are made. This also means that you may come across gaps in the documentation where little information is currently provided. Please do not hesitate to post on the forum if there is something missing that you need critically (or even not so critically).

Additionally, if you would like to contribute to the Speeduino documentation, we would love to hear from you! The preferred method to request wiki access is either via the forum (<https://speeduino.com/forum/viewforum.php?f=9>) or via Slack (<http://slack.speeduino.com:3000/>)

1.2 Connecting to TunerStudio

Tuner Studio is the recommended tuning interface for the Speeduino. It runs on Windows, Mac and linux and provides configuration, tuning and logging capabilities.

Once you have the firmware compiled and uploaded to your Arduino, you're ready to setup Tuner Studio in order to configure and monitor it. If you haven't yet compiled and uploaded the firmware, refer to the Compiling and Installing

Firmware page.

1.2.1 Downloading Tuner Studio

If you haven't already, grab a copy of Tuner Studio from: (<http://www.tunerstudio.com/index.php/downloads>) Tuner Studio is available for Windows, Mac and linux and will run on most PCs as it's system requirements are fairly low.

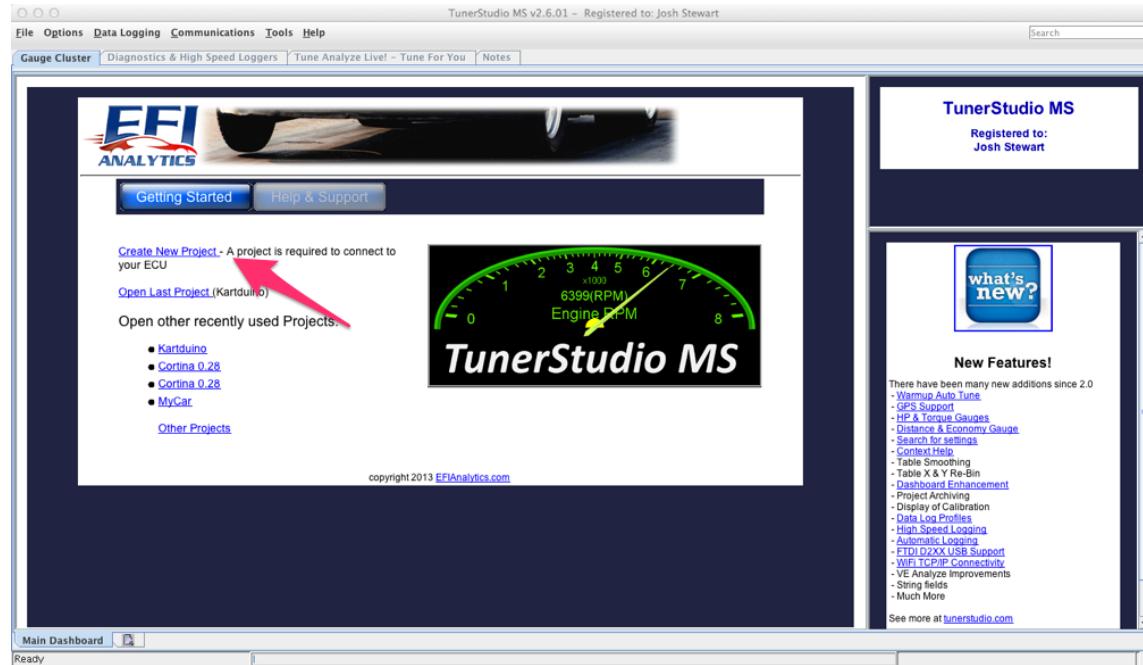
The current minimum version of TunerStudio required is 3.0.7, but the latest version is usually recommended.

If you find Tuner Studio to be useful, please consider paying for a license. This is a fantastic program from a single developer that rivals the best tuning software in the world, it's worth the money.

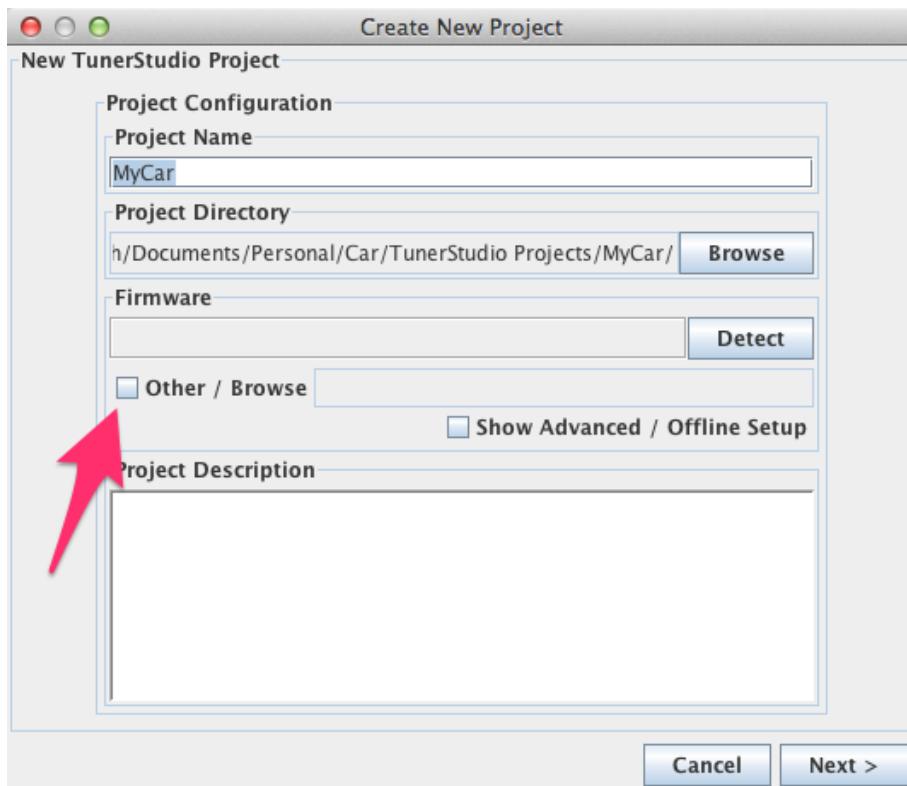
1.2.2 Setting up your project

Create new project

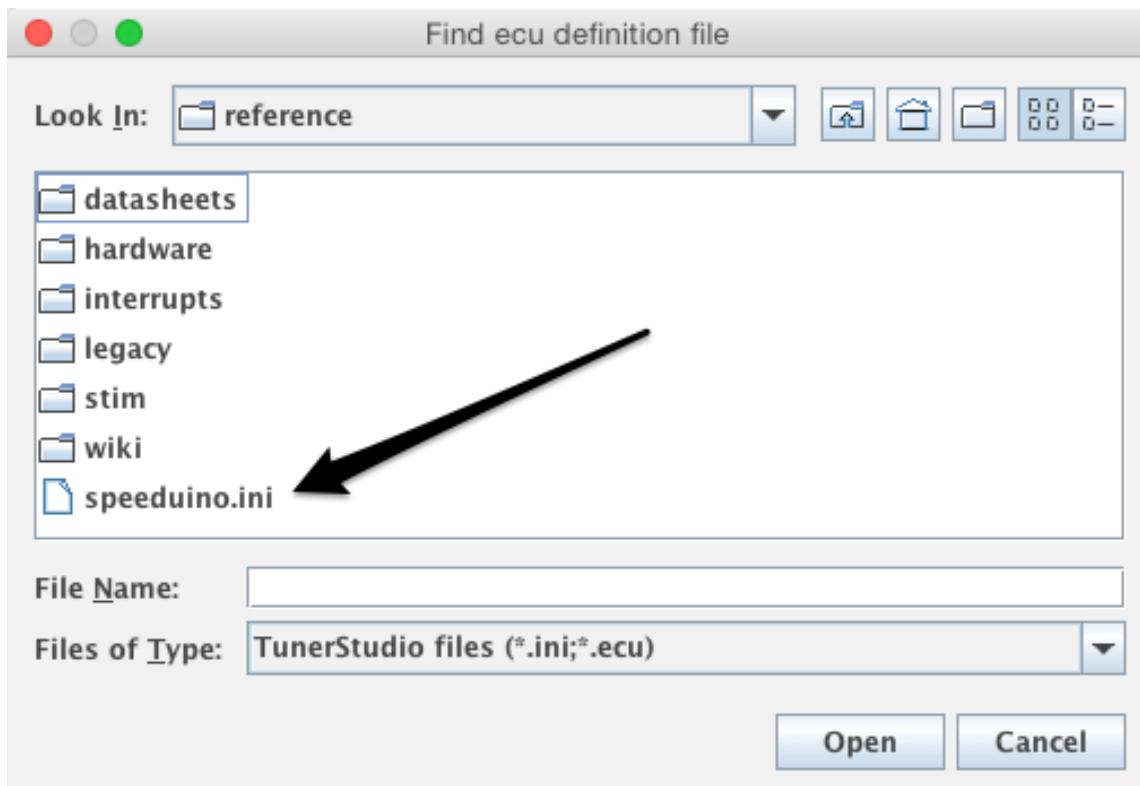
When you first start TunerStudio, you'll need to setup a new project which contains the settings, tune, logs etc. On the start up screen, select 'Create new project'



Give you project a name and select the directory you want the project to be stored in. Tuner Studio then requires a firmware definition file in order to communicate with the arduino. Tick the 'Other / Browse' button.

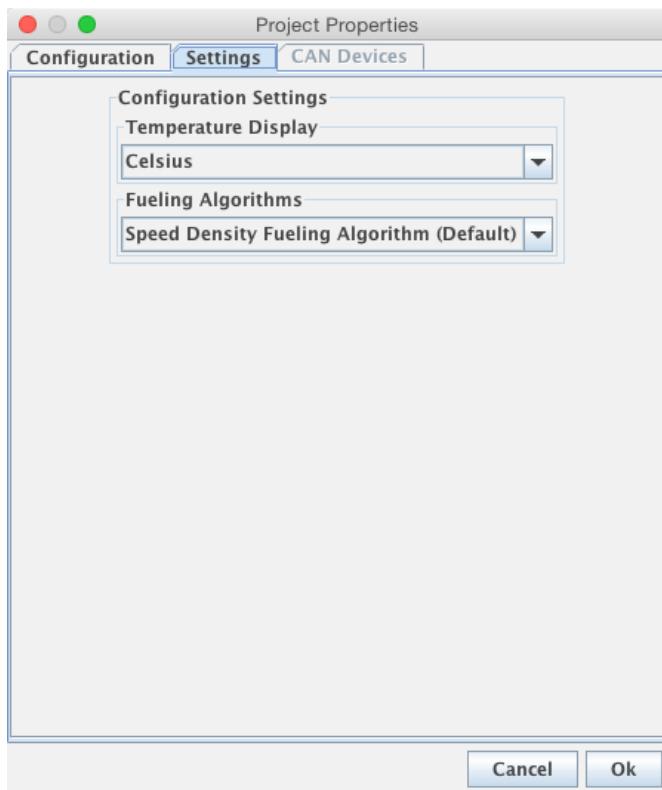


Then browse to the Speeduino source directory, enter the reference subfolder and select speeduino.ini file



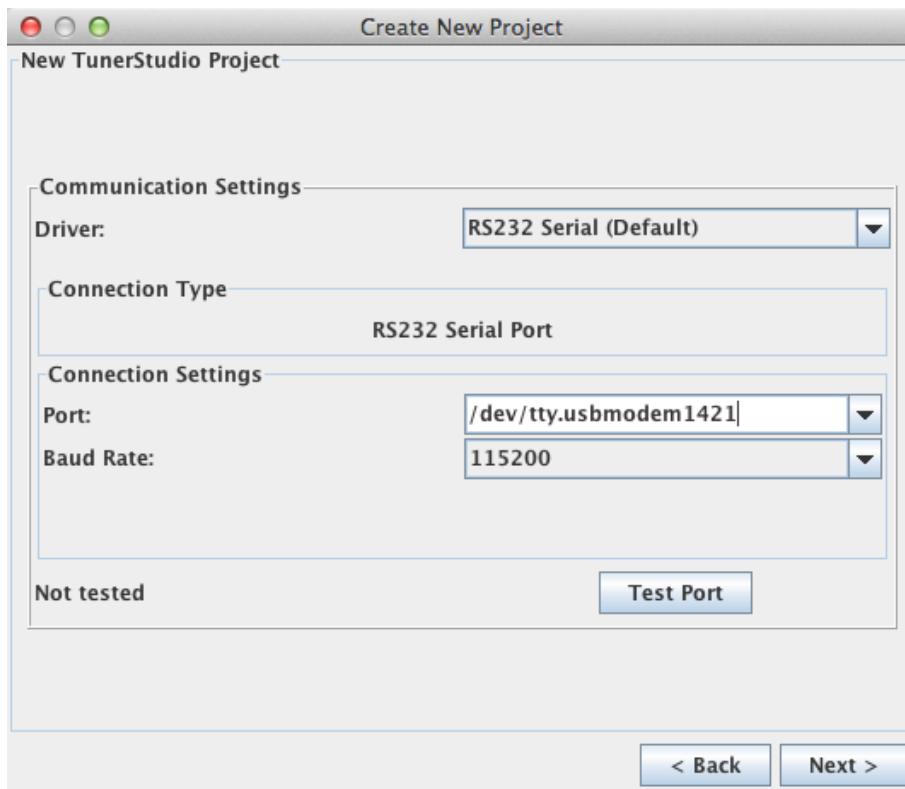
Configuration options

Set the configuration parameters for your project. These can be changed any time later on, so don't worry if you don't have them at this time.



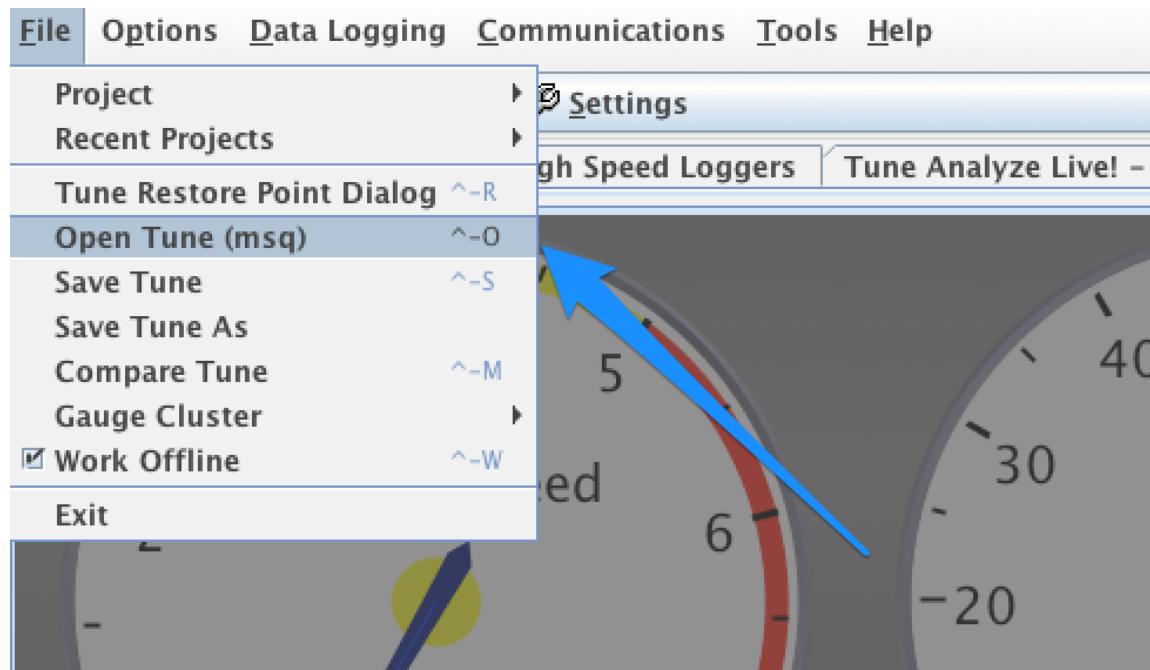
Comms settings

Select your comms options. The exact port name will depend on which operating system you are running and this will be the same as in the Arduino IDE. Baud rate should be 115200.

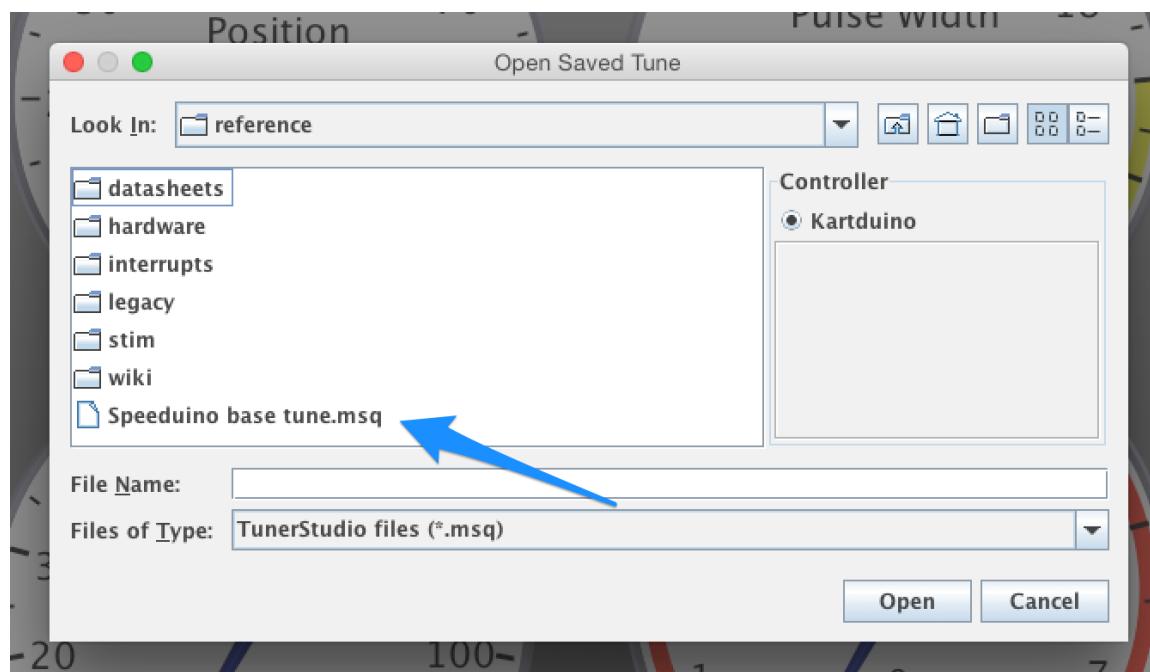


Load base tune

Once the project is created, you'll need to load in a base tune to ensure that all values are at least somewhat sane. Failure to do this can lead to very strange issues and values in your tune.



In the Speeduino reference directory, you will find the base tune file to be opened:

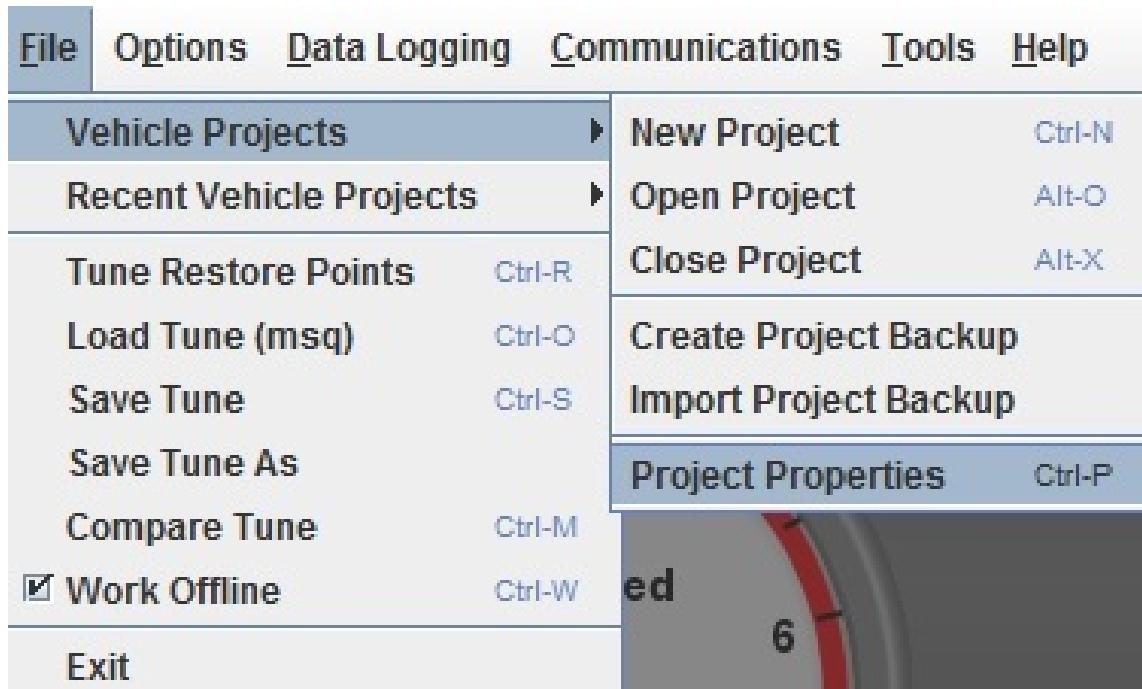


And that's it! Tuner Studio should now attempt to connect to the Arduino and show a realtime display of the ECU.

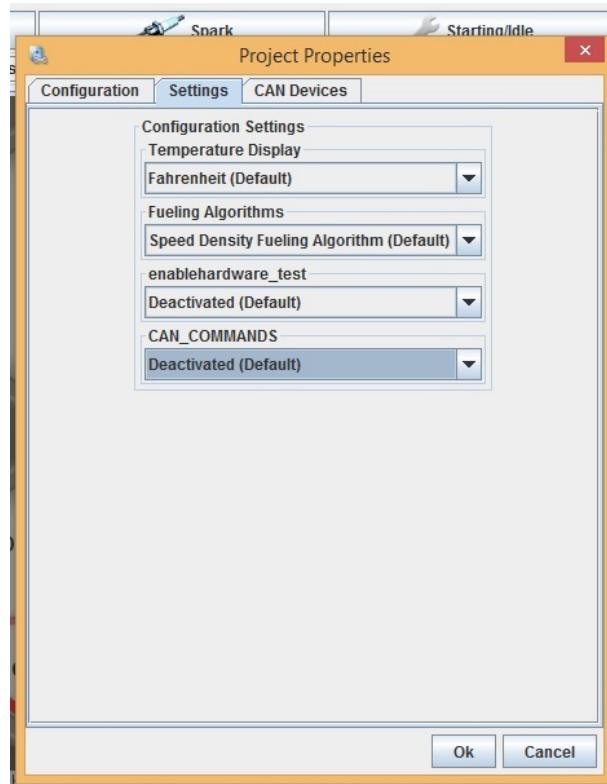
1.3 Creating a TunerStudio Project

1.3.1 Configuring TunerStudio Project Properties

The menu option for the project properties page can be found here



Once opened this page will be seen.



1.3.2 Settings Tab

Temperature Display

Options are :

- Fahrenheit(Default)
- Celsius

Fueling Algorithms

Options are :

- Speed Density Fueling Algorithm (default)
- Alpha-N Fueling Algorithm

Enable_hardware_test

Default option is disabled. If Enabled an additional Tab will appear on the tuning page



Clicking on this will open further options



- Output Testing

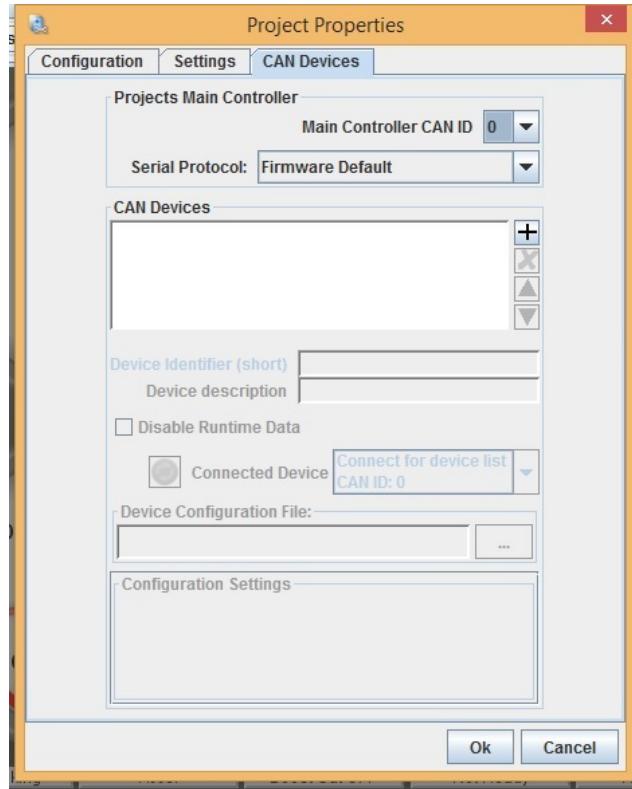
Hardware test page.

- Input Testing

CAN_COMMANDS

Default option is disabled

1.3.3 Can Devices Tab



Chapter 2

Hardware

2.1 MX5 PNP

2.1.1 Introduction

The Speeduino Miata / MX5 Plug N Play (PNP) box is designed for easy installation on the 1.6L NA6 vehicles using the 48-pin ECU. This is all 1.6L models from 1989 through 1993 and some 1.6s up to 1995.

The stock ECU for these vehicles have a 2 plug loom connection and look like the below:



WARNING: In particular, please see below in the fuel pump section for details that must be understood prior to starting

2.1.2 Hardware requirements

The PNP box plugs straight into the stock wiring in place of the original ECU, however some hardware changes are either recommended or are desirable in most installations.

Most significantly, Speeduino does not operate with the stock AFM on the NA6 engine. This unit can either be retained or removed, but if being kept in place, the connector to it should be disconnected.

Manifold Pressure

For a load reference, it is strongly recommended to run a manifold pressure line to the Speeduino PNP box. This allows Speeduino to run in the default Speed-Density configuration and is usually a fairly easy installation. The unit comes with a built-in MAP sensor that supports up to 1 Bar of boost, but is compatible with other external sensors if more pressure is required.

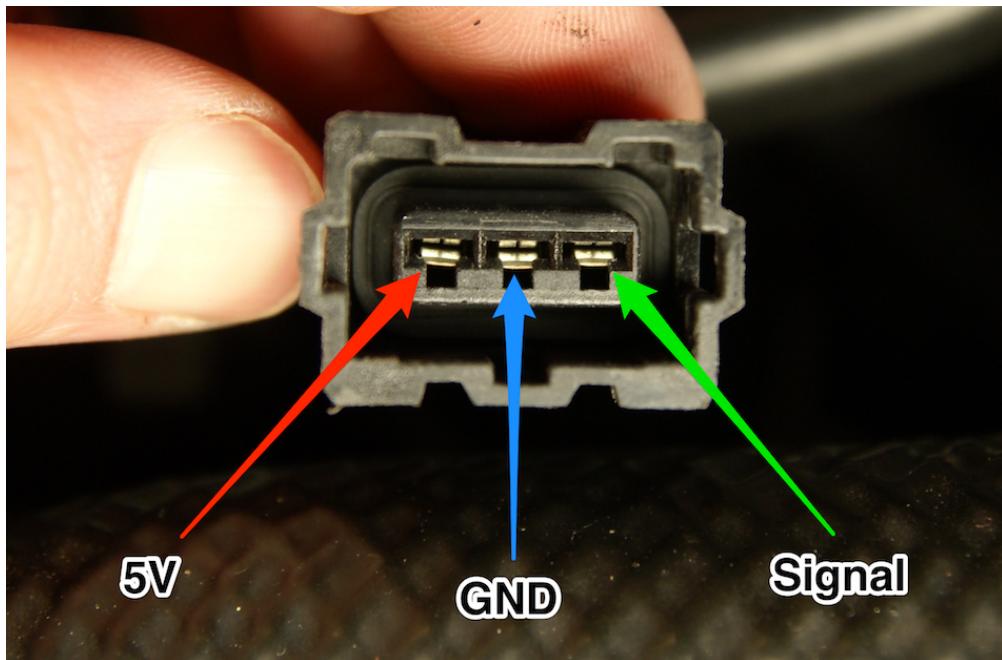
The 1.6L cars typically come with a suitable MAP port near the throttle body that is capped off in stock form and is generally the easiest place to take the manifold pressure reference.



5mm or 6mm vacuum hose should be used and there are multiple original holes in the firewall where this can be run.

Throttle Position Sensor

Manual NA6s come with a switch only TPS that provides limited feedback to the ECU. It is highly recommended to replace this with a Variable TPS (VTPS) that provides a signal indicating the current throttle position. The original wiring can be used with any 3 wire VTPS

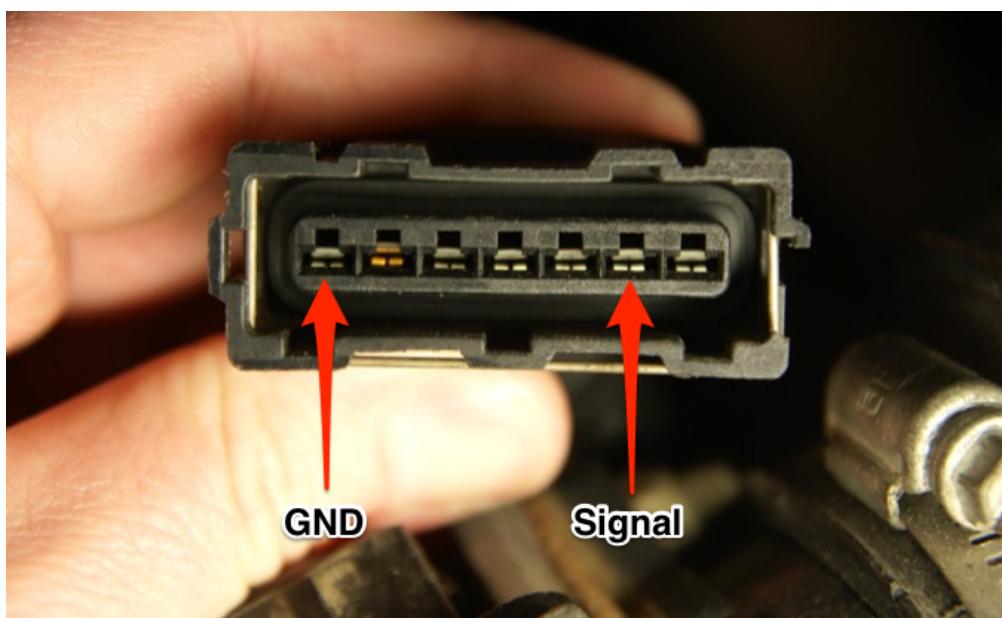


Note: If a VTPS is NOT being fitted, the stock TPS should be disconnected and a 1k resistor placed between the signal line and ground to prevent erratic acceleration enrichment at full throttle.

Inlet Temperature Sensor

In the stock configuration, inlet air temperature is provided by a sensor in the AFM. If the AFM is retained then this sensor will work if jump wires are run from the AFM to the disconnected connector (See image below), however as most setups elect to remove the AFM, an additional sensor needs to be added. The recommended sensor is the GM open air IAT that is common to many GM vehicles. Part number for this is #25036751 and it can be found fairly cheaply from many online sources, including the Speeduino store (https://speeduino.com/shop/index.php?id_product=23&controller=product)

The 2 wires from this sensor can be pinned directly into the AFM connector on pins 1 and 6 (It does not matter which wire goes to which pin):



Wideband O2 Sensor

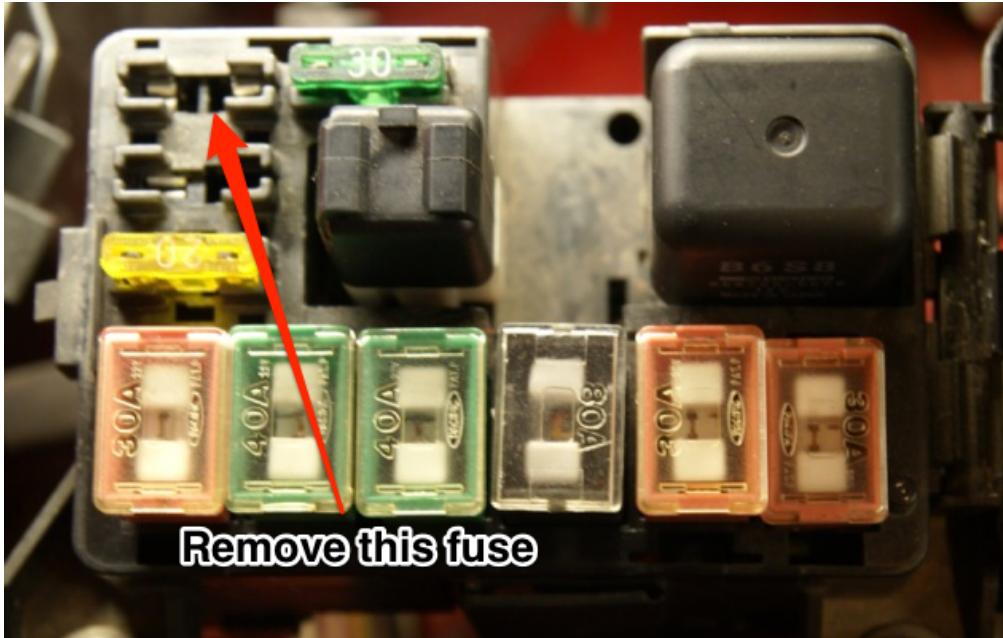
Whilst not mandatory, the installation of a wideband oxygen sensor and controller is strongly recommended. Any wideband controller that outputs a 0-5v signal is supported and calibration for common controllers can be found in the Tools->Calibrate AFR Table dialog.

The wideband analog output signal should be connected to the original O2 sensor wire. This has a convenient connector in the engine bay, located just next to the coils. This can be found by following the wire from the original sensor. If not reusing the original narrowband sensor, the connector can be cut from this and attached to the wideband signal.

Fuel pump control

The stock ECU does not perform any fuel pump control as this is taken care of by the AFM. Speeduino however can control the fuel pump through the original wiring, but requires the removal of the ST_SIG fuse. Failure to remove this fuse prior to powering the unit on will trip the smart FET that is used on this line, but should not cause permanent damage if only performed once or twice.

The fuse to be removed is found in the engine bay fuse block:



Alternative control methods If the above method of fuel pump control is either not desirable or not available, an alternative driver wired to pin 2O on the main connector that can be used for this. Pin 2O originally carries the AFM signal, however as Speeduino does not use this (And the AFM must be disconnected) it can be used to carry the fuel pump control.

To do this, a jumper wire is required on the AFM connector per the below:



Once the above jumper is in place, the fuel pump pin in TunerStudio should be set to A9.

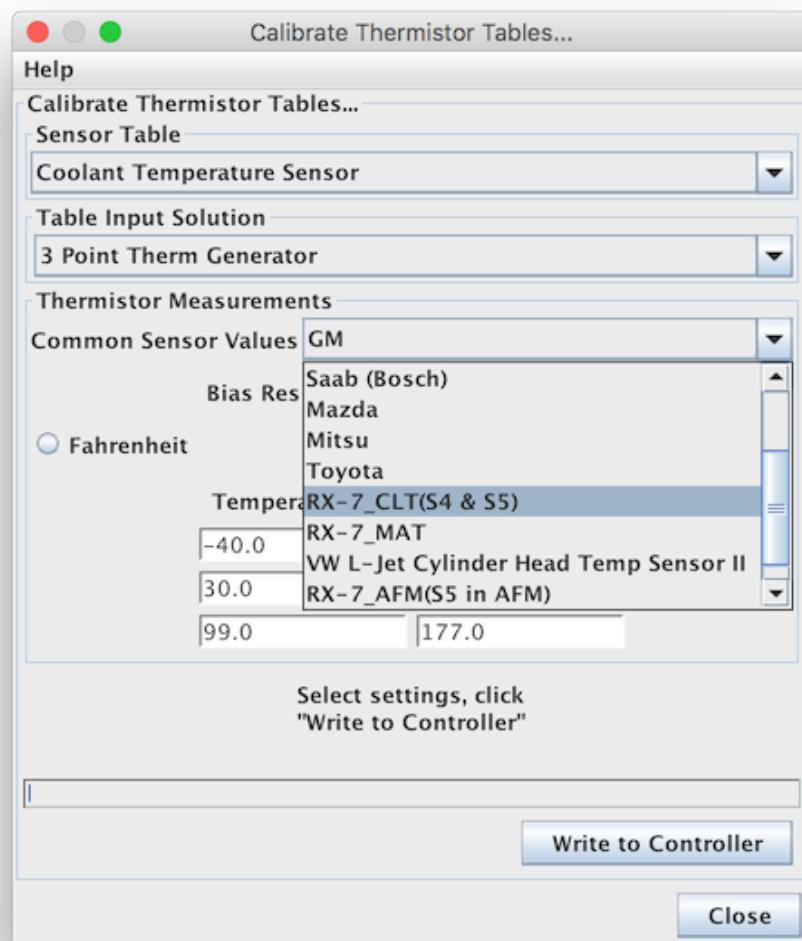
2.1.3 Configuration and start

Sensor calibration

The stock sensors can use preset calibrations within TunerStudio. The following values should be used if the stock sensors are retained:

Stock Coolant Sensor (CLT) - RX-7_CLT(S4 & S5)

Stock inlet air sensor (IAT) - RX-7_AFM(S5 in AFM)



Chapter 3

Configuration

3.1 Engine Constants

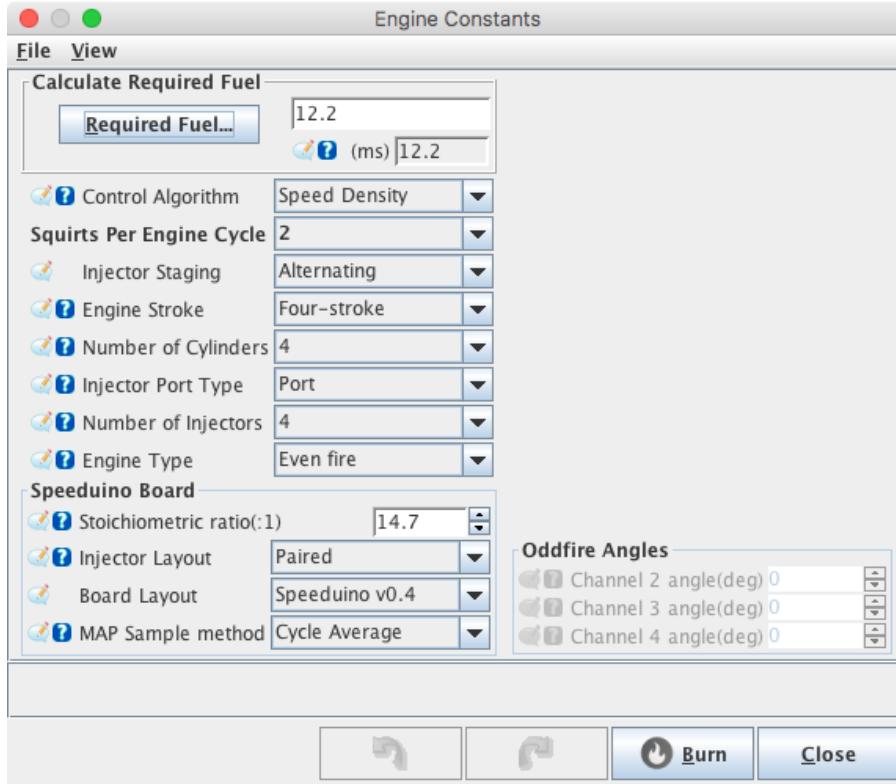
3.1.1 Overview[edit]

From the Settings menu, select Constants



Here you need to setup the engine constants. Fill out the fields in the bottom section before calculating the Required Fuel.

3.1.2 Settings[edit]



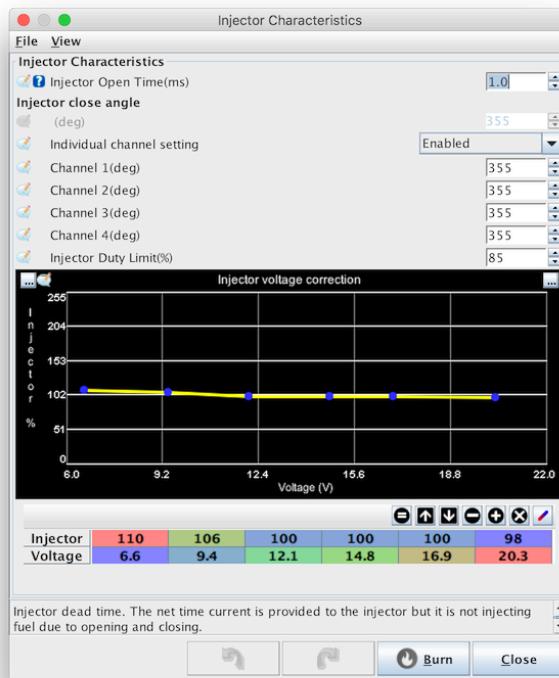
- **Control Algorithm:** The load source that will be used for the fuel table
- **Squirts per Engine Cycle:** How many squirts will be performed over the duration of the engine cycle (Eg 720 degrees for a 4 stroke). most engines will not require values greater than 4. For sequential installations, this should be set to 2 with the injector staging set to 'Alternating'(Internally Speeduino will adjust the squirts to 1)
- **Injector Staging:** This configures the timing strategy used for the injectors
 - Alternating (Recommended for most installs) - Injectors are timed around each cylinders TDC. The exact closing angle can be specific in the Injector Characteristics dialog.
 - Simultaneous - All injectors are fired together, based on the TDC of cylinder 1.
- **Engine stroke:** Whether the engine is 2 stroke or 4 stroke
- **Number of cylinders:** Number of cylinders in the engine. For rotary engines, select 4.
- **Injector Port Type:** Option isn't used by firmware. Selection currently does not matter
- **Number of injectors:** Usually the same as number of cylinders (For port injection)
- **Engine Type:** Whether the crank angle between firings is the same for all cylinders. If using an Odd fire engine (Eg Some V-Twins and Buick V6s), the angle for each output channel must be specific.
- **Injector Layout:** Specifies how the injectors are wired in
 - **Paired:** 2 injectors are wires to each channel. The number of channels used is therefore equal to half the number of cylinders.
 - **Semi-Sequential:** Semi-sequential: Same as paired except that injector channels are mirrored (1&4, 2&3) meaning the number of outputs used are equal to the number of cylinders. Only valid for 4 cylinders or less.
 - **Sequential:** 1 injector per output and outputs used equals the number of cylinders. Injection is timed over full cycle. Only available for engines with 4 or fewer cylinders.

- **Board Layout:** Specifies the input/output pin layout based on which Speeduino board you're using. For specific details of these pin mappings, see the utils.ino file
- **MAP Sample Method:** How the MAP sensor readings will be processed:
 - **Instantaneous:** Every reading is used as it is taken. Makes for a highly fluctuating signal, but can be useful for testing
 - **Cycle Average:** The average sensor reading across 720 crank degrees is used. This is the recommended option for 4 or more cylinders
 - **Cycle Minimum:** The lowest value detected across 720 degrees is used. This is the recommended method for less than 4 cylinders or ITBs

3.2 Injector Characteristics

3.2.1 Overview

3.2.2 Settings

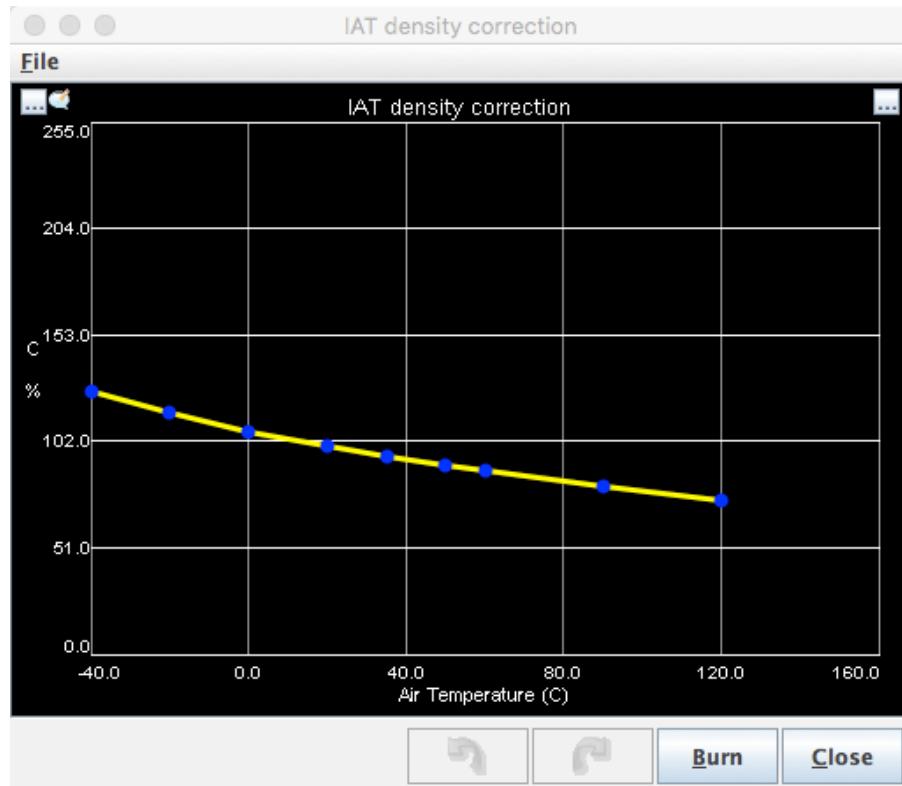


3.3 IAT Density

3.3.1 Overview

The IAT density curve represents the change in oxygen density of the inlet charge as temperature rises. The default curve approximately follows the ideal gas law and is suitable for most installations, however if you are seeing very high inlet temperatures (Either due to heat soak in the engine bay or from turbocharging) the you may need to adjust the hot end of this curve.

3.3.2 Settings



3.4 Acceleration Enrichment

3.4.1 Overview

Acceleration Enrichment (AE) is used to add extra fuel during the short transient period following a rapid increase in throttle. It performs much the same function as an accelerator pump on a carbureted engine, increasing the amount of fuel delivered until the manifold pressure reading adjusts based on the new load.

To operate correctly, you must have a variable TPS installed and calibrated.

3.4.2 Theory

Tuning of acceleration enrichment is based on the rate of change of the throttle position, a variable known as TPSdot (TPS delta over time). This is measured in %/second, with higher values representing faster presses of the throttle and values in the range 50%/s to 1000%/s are normal. Eg:

- 100%/s = pressing the throttle from 0% to 100% in 1 second
- 1000%/s = pressing the throttle from 0% to 100% in 0.1s

TPSdot forms the X axis of the acceleration curve, with the Y axis value representing the % increase in fuel.



Tuning

The enrichment curve included with the base Speeduino tune is a good starting point for most engines, but some adjustment is normal depending on injector size, throttle diameter etc.

In most cases, tuning of the AE curve can be performed in a stationary environment, though dyno or road tuning is also possible. Fast and slow blips of the throttle should be performed and the affect on the AFRs monitored using the live line graph on the AE dialog. This graph shows both TPSdot and AFR values in sync with each other, making adjustments to the correct part of the AE curve simpler to identify.

If you find that the AFR is initially good, but then goes briefly lean, you should increase the 'Accel Time' setting, with increments of 10-20ms recommended.

False triggering In cases where the TPS signal is noisy, spikes in its reading may incorrectly trigger the acceleration enrichment. This can be seen in a log file or on a live dash in TunerStudio by the activation of the 'TPS Accel' indicator when there is no (or little) throttle movement occurring. Should this occur (and assuming that the TPS wiring cannot be corrected to reduce noise) then the false triggers can be prevented from triggering AE by increasing the "TPSdot Threshold" value. This should be increased in increments of ~5%/s, pausing between each increase to observe whether AE is still being incorrectly activated.

3.5 Rev Limits

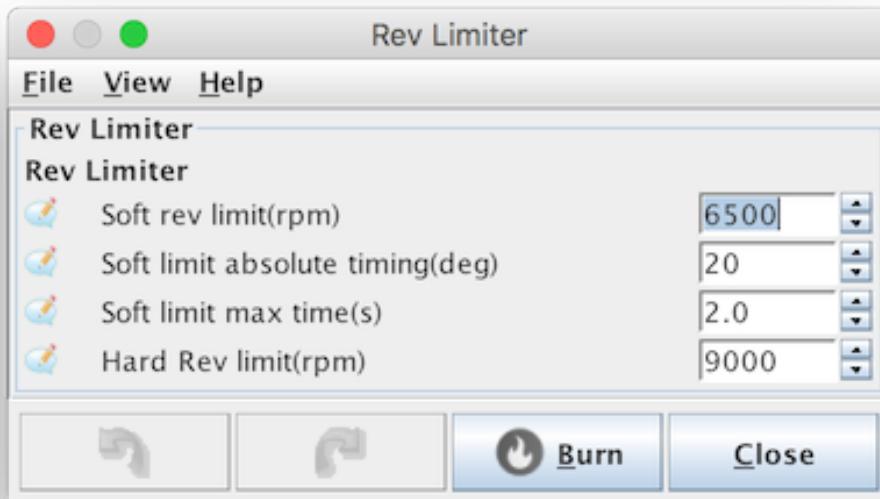
3.5.1 Overview

Speeduino includes a spark based rev limited with both hard and soft cuts.

The soft cut limiter will lock timing at an absolute value to slow further acceleration. If RPMs continue to climb and reach the hard cut limit, ignition events will cease until the RPM drop below this threshold.

Note As this is spark based limiting, fuel only installs cannot use the rev limiter functionality

3.5.2 Settings



- **Soft rev limit:** The RPM at which the soft cut ignition timing will be applied over.
- **Soft limit absolute timing:** Whilst the engine is over the soft limit RPM, the ignition advance will be held at this value. Lower values here will have a greater soft cut affect.
- **Soft limit max time:** The maximum number of seconds that the soft limiter will operate for. If the engine remains in the soft cut RPM region longer than this, the hard cut will be applied.
- **Hard rev limiter:** Above this RPM, all ignition events will cease.

3.6 Flex Fuel

3.6.1 Overview

Speeduino has the ability to modify fuel and ignition settings based on the ethanol content of the fuel being used, a practice typically known as flex fuelling. A flex fuel sensor is installed in the feed or return fuel lines and a signal wire is used as an input on the Speeduino board.

As ethanol is less energy dense, but also has a higher equivalent octane rating, adjustments to the fuel load and ignition timing are required.

3.6.2 Hardware

Speeduino uses any of the standard GM/Continental Flex fuel sensors that are widely available and were used across a wide range of vehicles. These were available in 3 different units, all of which are functionally identical, with the main difference being only the physical size and connector. The part numbers for these are:

- Small - #13577429
- Mid-size - #13577379

- Wide - #13577394 (Same as the mid-size one, but with longer pipes)

All 3 use a variant of the Delphi GT150 series connector. You can use a generic GT150 connector, but you will have to clip off 2 tabs from the side of the sensor.

Part numbers :

- Housing (#13519047)
- Pins (#15326427)
- Seal (#15366021)

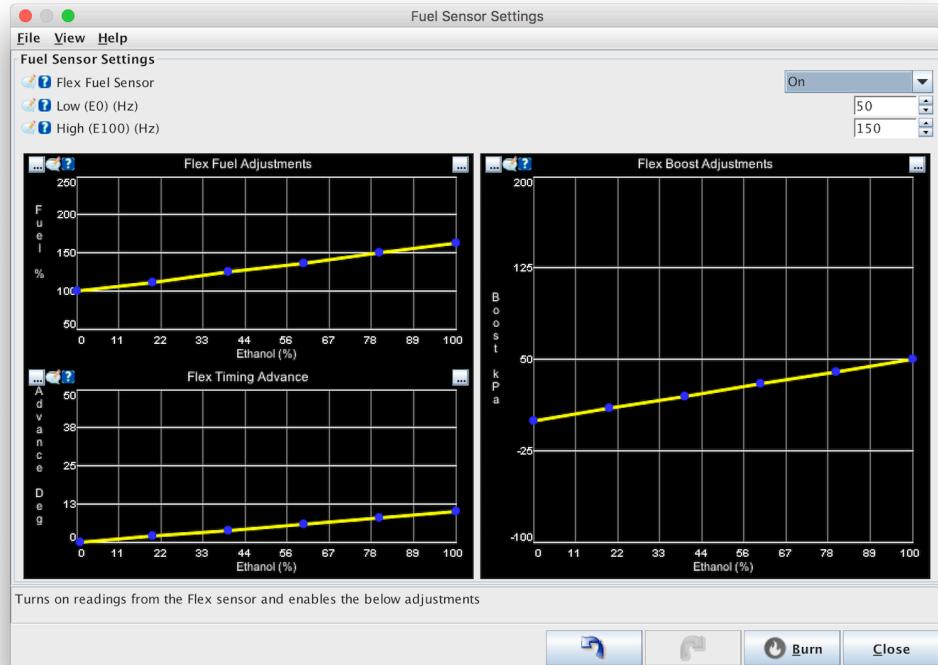
Alternatively, there is a GM part for a harness connector, part number 13352241: <http://www.gmpartsdirect.com/oe-gm/13352241>

Wiring

All units are wired identically and have markings on the housing indicating what each pin is for (12v, ground and signal) Speeduino boards v0.3.5+ and v0.4.3+ have an input location on their proto areas that the signal wire can be directly connected to.

On boards earlier to these, you will need to add a pullup resistor of between 2k and 3.5k Ohm. Recommended value is 3.3k, however any resistor in this range will work. Note that this is a relatively strict range, more generic values such as 1k or 10k DO NOT WORK with these sensors.

3.6.3 Tuning



- **Sensor frequency** - The minimum and maximum frequency of the sensor that represent 0% and 100% ethanol respectively. For standard GM/Continental flex sensors, these values are 50 and 150

- **Fuel multiplier%** - This is the additional fuel that should be added as ethanol content increases. The Low value on the left represents the adjustment to the fuel map at 0% ethanol and will typically be 100% if the base tune was performed with E0 fuel. If the base tune was made with E10 or E15 however, this value can be adjusted below 100%. The high value represents the fuel multiplier at 100% ethanol (E100) and the default value of 163% is based on the theoretical difference in energy density between E0 and E100. Tuning of this value may be required
- **Additional advance** - The additional degrees of advance that will be applied as ethanol content increases. This amount increases linearly between the low and high values and is added after all other ignition modifiers have been applied.

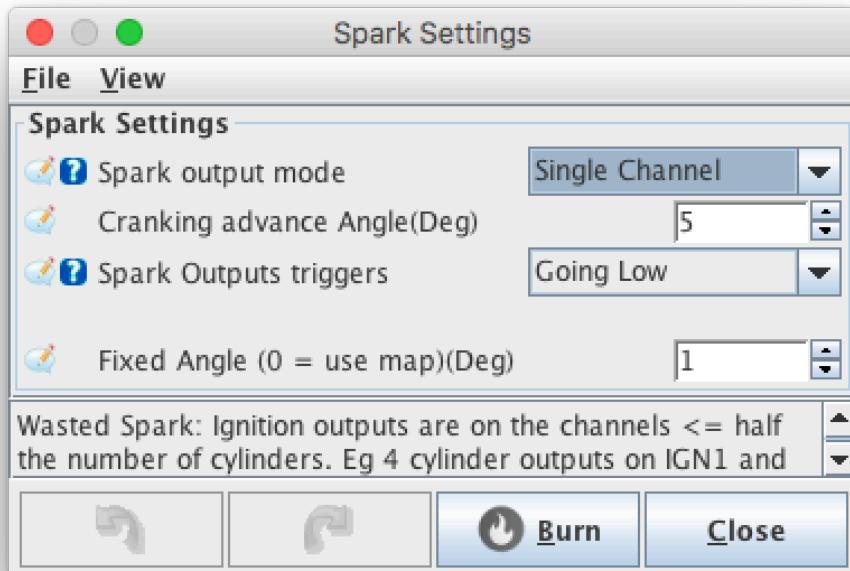
3.7 Spark Settings

3.7.1 Overview

The Spark settings dialog contains the options for how the ignition outputs will function, including which of the 4 IGN outputs are used and how. They are critical and incorrect values will result in an engine not starting and in some cases damage to hardware is possible. This dialog also contains a number of options for fixing the ignition timing for testing and diagnosis.

Please ensure you have reviewed these settings prior to attempting to start your engine.

3.7.2 Settings



- **Spark Output mode** - Determines how the ignition pulses will be outputted and is very specific to your ignition wiring. **Note that no matter which option is selected here, ignition signals ALWAYS fire in numerical order (ie 1->2->3->4) up to the maximum number of outputs.** The firing order of the engine is accounted for in the wiring order.

- **Wasted Spark** - Number of ignition outputs is equal to have the number of cylinders and each output will fire once every crank revolution. One spark will therefore take place during the compression stroke and the other on the exhaust stroke (aka the 'wasted' spark). This method is common on many 80s and 90s vehicles that came with specific wasted spark coils, but can also be used with individual coils that are wired in pairs. Wasted spark will function with only a crank angle reference (Eg a missing tooth crank wheel with no cam signal)
 - **Single Channel** - This mode sends all ignition pulses to IGN1 output and is used when the engine contains a distributor (Typically with a single coil). The number of output pulses per (crank) revolution is equal to half the number of cylinders.
 - **Wasted COP** - This is a convenience mode that uses the same timing as the 'Wasted Spark' option, however each pulse is sent to 2 ignition outputs rather than one. These are paired IGN1/IGN3 and IGN2/IGN4 (ie When IGN1 is high, IGN3 will also be high). As this is still a wasted spark timing mode, only crank position is required and there will be 1 pulse per pair, per crank revolution. This mode can be useful in cases where there are 4 individual coils, but running full sequential is either not desired or not possible (Eg when no cam reference is available).
 - **Sequential** - This mode is only functional on engines with 4 or fewer cylinders.
 - **Rotary** - See below for full detail
- **Cranking advance** - The number of absolute degrees (BTDC) that the timing will be set to when cranking. This overrides all other timing advance modifiers during cranking.
 - **Spark output triggers** - **THIS IS A CRITICAL SETTING!**. Selecting the incorrect option here can cause damage to your igniters or coils. Specifies whether the coil will fire when the ignition output from Speeduino goes HIGH or goes LOW. The VAST majority of ignition setups will require this to be set **GOING LOW** (ie the coil charges/dwells when the signal is high and will **fire** when that signal goes low). Whilst GOING LOW is required for most ignition setups, there are some configurations that perform the dwell timing on the ignition module and fire the coil only when they receive a HIGH signal from the ECU.
 - **Fixed Angle** - This is used to lock the ignition timing to a specific angle for testing. Setting this to any value other than 0 will result in that exact angle being used (ie overriding any other settings) at all RPMs/load points, except during cranking (Cranking always uses the above Cranking Advance setting). This setting should be set to 0 for normal operation.

Rotary modes



Speeduino currently only supports the ignition configuration used on FC RX7 engines. Support for FD and RX8 ignition setups is in development. The leading / trailing split angle can be set as a function of the current engine load.

- **FC** - Outputs are configured for the Leading/Trailing setup that was used on FC RX7s. Wiring required is:
 - **IGN1** - Leading spark
 - **IGN2** - Trailing spark
 - **IGN3** - Trailing select
- **FD** - Not currently supported
- **RX8** - Not currently supported

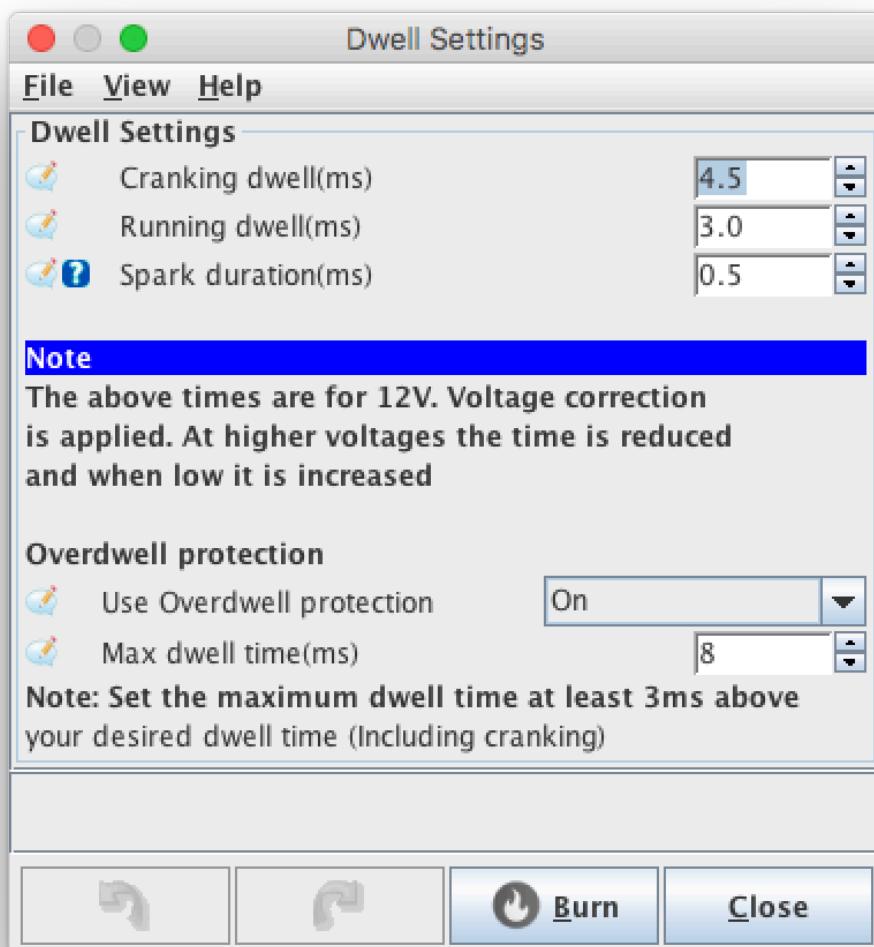
3.8 Dwell

3.9 Overview

The dwell control dialog alters the coil charging time (dwell) for Speeduino's ignition outputs. Care should be taken with these settings as igniters and coils can be permanently damaged if dwelled for excessive periods of time.

From the April 2017 firmware onwards, dwell will automatically reduce when the configured duration is longer than the available time at the current RPM. This is common in single channel ignition configurations (Eg 1 coil with a distributor) and in particular on higher cylinder count engines.

3.10 Settings



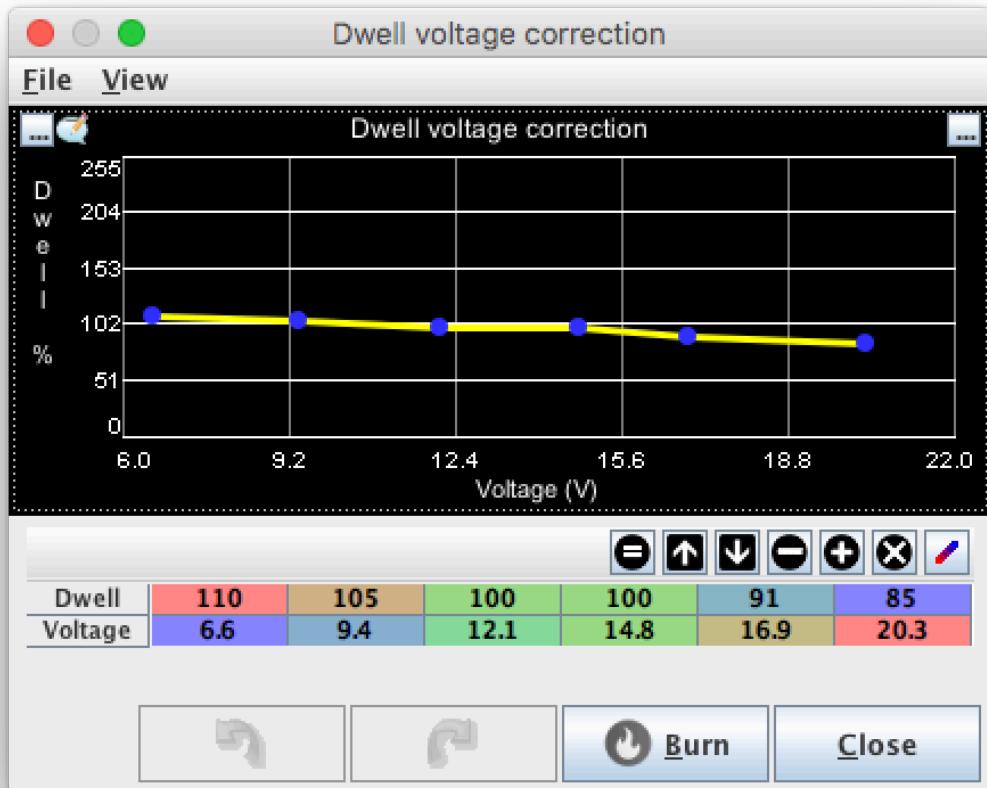
Note: Both the running and cranking dwell times are nominal values, assumed to be at a constant voltage (Usually 12v). Actual dwell time used will depend on the current system voltage with higher voltages having lower dwell times and vice versa. See section below on voltage correction

- Cranking dwell - The nominal dwell time that will be used during cranking. Cranking is defined as being whenever the RPM is above 0, but below the 'Cranking RPM' values in the Cranking dialog
- Running dwell - The nominal dwell that will be used when the engine is running normally.
- Spark duration - The approximate time the coil takes to fully discharge. This time is used in calculating a reduced dwell when in time limited conditions, such as mentioned above on single coil, high cylinder count engines. The limited dwell time is calculated by taking the maximum revolution time at the given RPM, dividing by the number of spark outputs required per revolution and subtracting the spark duration. Outside of those conditions, this setting is not used.
- Over dwell protection - The over dwell protection system runs independently of the standard ignition schedules and monitors the time that each ignition output has been active. If the active time exceeds this amount, the output will be ended to prevent damage to coils. This value should typically be at least 3ms higher than the nominal dwell times configured above in order to allow overhead for voltage correction.

3.10.1 Voltage correction

As the system voltage rises and falls, the dwell time needs to reduce and increase respectively. This allows for a consistent spark strength without damaging the coil/s during high system voltage conditions. It is recommended that 12v be used as the 'nominal' voltage, meaning that the Dwell % figure at 12v should be 100%.

The correction curve in the base tune file should be suitable for most coils / igniters, but can be altered if required.

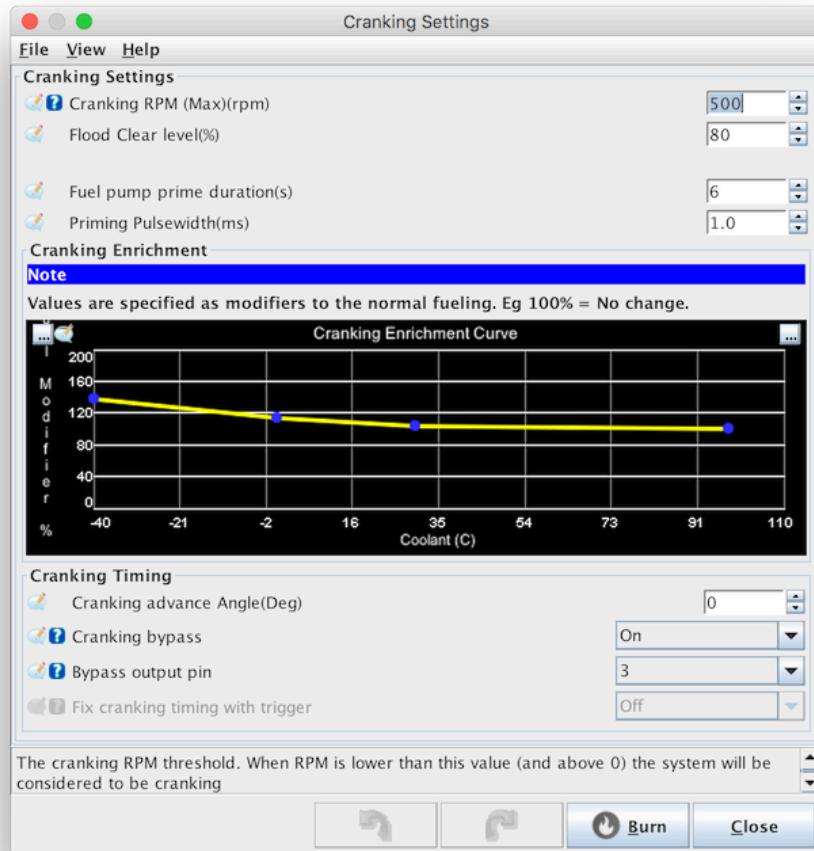


3.11 Cranking

3.11.1 Overview

Cranking conditions during starting typically require multiple adjustments to both fuel and ignition control in order to provide smooth and fast starts. The settings on this dialog dictate when Speeduino will consider the engine to be in a cranking/startup condition and what adjustments should be applied during this time.

3.11.2 Settings



- **Cranking RPM** - This sets the threshold for whether Speeduino will set its status to be cranking or running. Any RPM above 0 and below this value will be considered cranking and all cranking related adjustments will be applied. It's generally best to set this to be around 100rpm higher than your typical cranking speed to account for spikes and to provide a smoother transition to normal idle
- **Flood Clear level** - Flood clear is used to assist in removing excess fuel that has entered the cylinder/s. Whilst flood clear is active, all fuel and ignition events will be stopped and the engine can be cranked for a few seconds without risk of starting or further flooding. To trigger flood clear, the RPM must be **below** the above Cranking RPM setting and the TPS must be **above** the threshold of this setting.
- **Fuel pump prime duration** - When Speeduino is first powered on, the fuel pump output will be engaged for this many seconds in order to pressurise the fuel system. If the engine is started in this time, the pump will simply keep running, otherwise it will be turned off after this period of time. Note that fuel pump priming only occurs at system power on time. If you have USB connected, Speeduino remains powered on even without a 12v signal.
- **Priming Pulsewidth** - Upon power up, Speeduino will fire all injectors for this period of time. This pulse is NOT intended as a starting fuel load, but is instead for clearing out air that may have entered the fuel lines. It should be kept short to avoid engine flooding.
- **Cranking enrichment** - Whilst cranking is active (See Cranking RPM above), the fuel load will be increased by this amount. Note that as a standard correction value, this cranking enrichment **is in addition** to any other adjustments that are currently active. This includes the warmup enrichment etc.
- **Cranking Bypass** - This option is specifically for ignition systems that have a hardware cranking ignition option. These systems were used throughout the 80s and early 90s and allowed ignition timing to be fixed and

controlled by the ignition system itself when active (Via an input wire). With this option you can specify an output pin that will be set HIGH when the system is cranking. The pin number specified is the ARDUINO pin number.

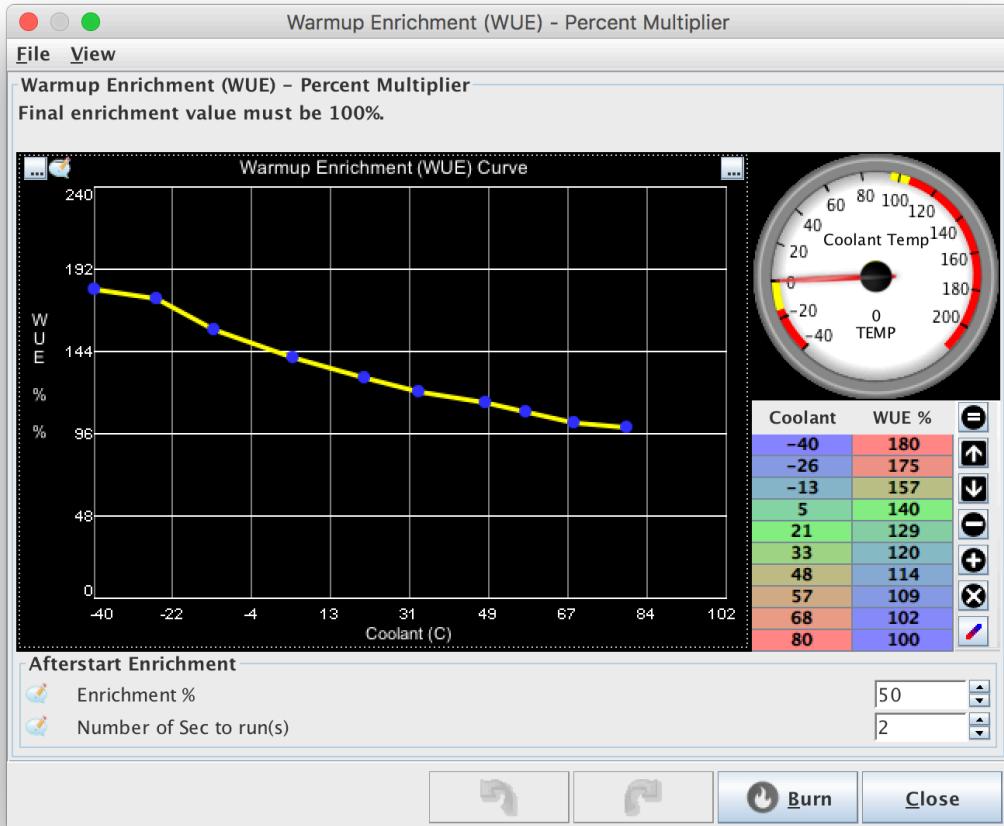
- **Fix cranking timing with trigger** - Some (usually low resolution) trigger patterns are designed to align one of their pulses with the desired cranking advance. This is typically 5 or 10 degrees BTDC. When enabled, Speeduino will wait for this timed input pulse before firing the relevant ignition output (A dwell safety factor is still applied incase this pulse is not detected). This option is only made available when a trigger pattern that supports this function is selected (See Trigger Setup)

3.12 Warmup

3.12.1 Overview

The Warm Up Enrichment (WUE) dialog contains settings related to the period after start (ie not cranking) but before the engine has reached normal operating temperature. It allows for modifications to fueling during this time to

3.12.2 Settings



Warmup curve

This curve represents the additional fuel amount to be added whilst the engine is coming up to temperature (Based on the coolant sensor). The final value in this curve should represent the normal running temperature of the engine

and have a value of 100% (Representing no modification of the fuel from that point onwards).

Afterstart Enrichment

Afterstart Enrichment (ASE) is a separate fuel modifier that operates over and above the WUE for a fixed period of time after the engine first starts. Typically this is a 3 - 10 second period where a small enrichment can help the engine transition smoothly from cranking to idling.

3.13 Idle

3.13.1 Overview

Compatible Idle Valve Types

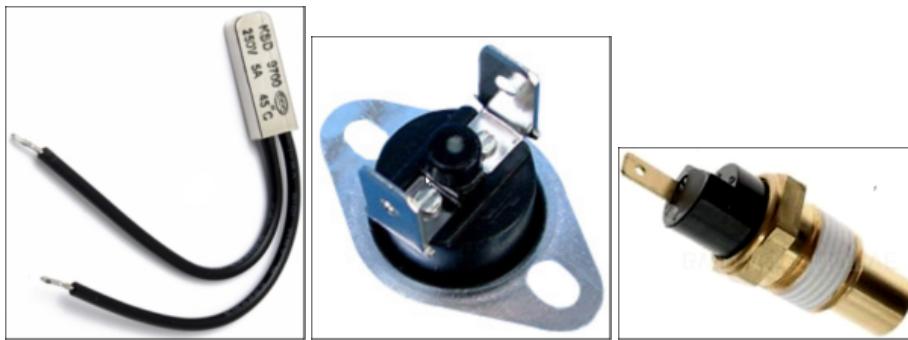
There are currently 3 modes of idle control available, using on/off, PWM duty cycle, or a stepper step count, enabled below a set coolant temperature. These modes cover the most common types of idle mechanisms in use. At this time only open loop control is available, meaning that an air bypass passage is enabled, rather than a target RPM. Closed loop control is anticipated at some point in the future, but no commitment is currently made.

Stand-Alone (Non-Electronic)

While not an idle control mode, Speeduino is compatible with stand-alone idle valves that are self-controlling. Examples of this are thermal wax or bi-metal spring idle or auxiliary air valves like the one below. Internally expanding and contracting material opens and closes air valves, providing increased air flow and engine rpm when cold for warmup. Speeduino functions to enrich the cold engine and adjust for the additional air, in the same way it would if you opened the throttle slightly.



Other examples of stand-alone valves are simple On/Off valves as shown in the next section, controlled by inexpensive thermal switches like these:



On/Off

This is a simple digital on/off “switch” output by Speeduino that triggers at a selected temperature. It is intended to control an on/off fast idle valve as found in many older OEM setups, or an open/closed solenoid-type valve that is chosen for the purpose. In addition to OEM idle valves, examples of valves popular for re-purposing as on/off idle valves are larger vacuum, breather, or purge valves, and even fuel valves. Idle speed adjustment is generally set only once, with an in-line adjustable or fixed restrictor, pinch clamp, or other simple flow-control method.



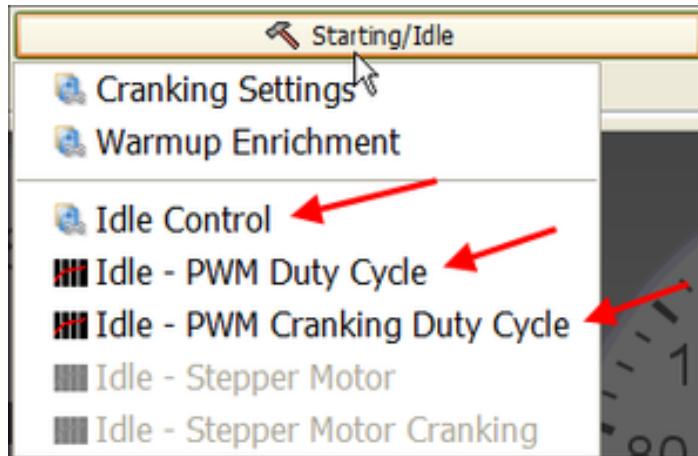
Note: On/Off valves can be used in many ways to increase or decrease air flow for various idle purposes in-addition to warm-up. Examples are use as dashpot valves to reduce deceleration stalling, idle speed recovery for maintaining engine speed with accessory loads such as air conditioning, or air addition for specific purposes such as turbo anti-lag air control. See [Generic Outputs](#) for control information.

PWM

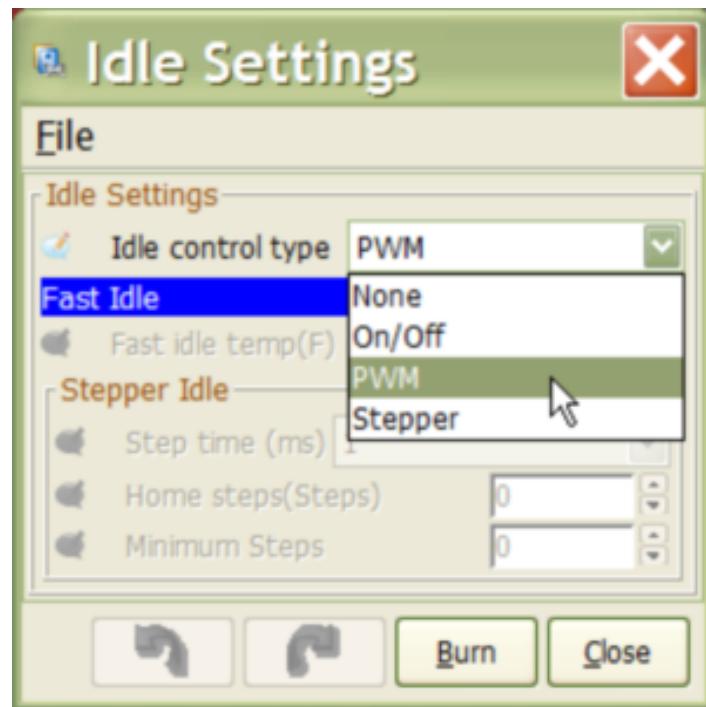
While similar in construction to many solenoid on/off valves; PWM idle valves are designed to vary the opening, and therefore flow through the valve, by PWM valve positioning.

Open-loop Duty Cycle Control Speeduino currently operates PWM valves in open-loop, effectively creating an on/off valve with adjustable flow. While the PWM duty cycle (DC) adjustment affects engine rpm, the adjustment is for valve opening, and therefore airflow and rpm are affected differently under various conditions. Note some idle valves default with no PWM signal to the open position, others closed, and some partially-open that close then re-open with increasing PWM DC. Be sure to research or test your valve type for proper operation.

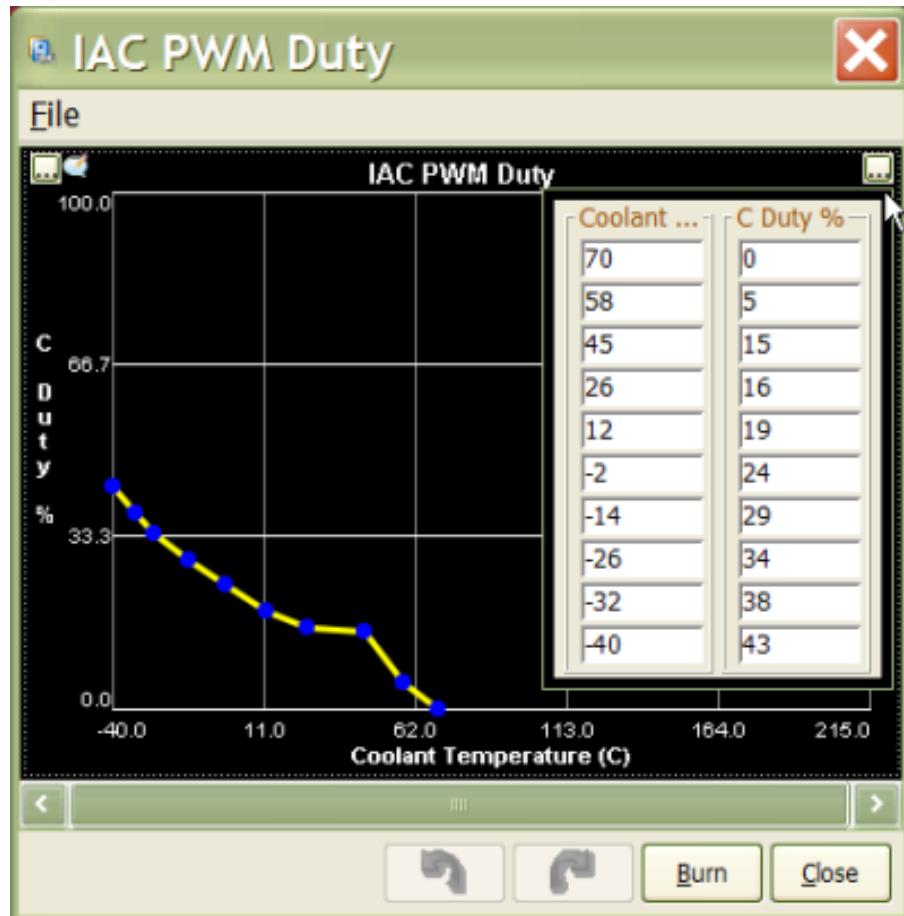
PWM Settings Settings in TunerStudio include selecting PWM idle control, temperature and DC settings for warmup, and PWM DC during cranking under the following selections:



Under Idle control type, PWM is selected:

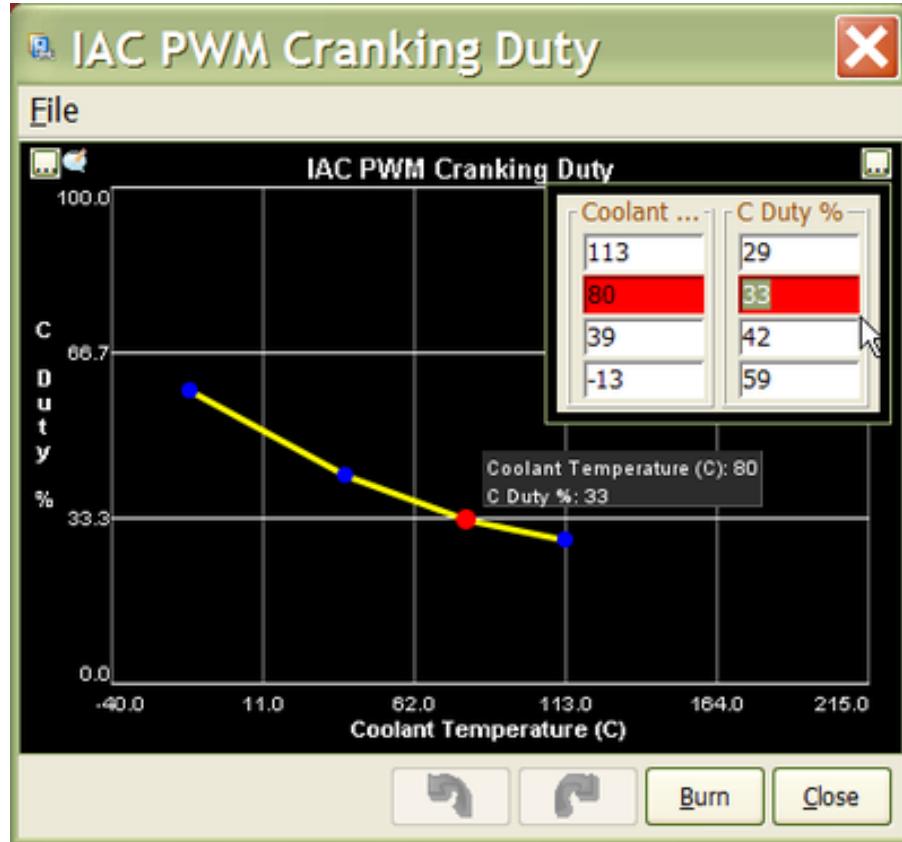


The temperature-versus-DC is selected under the Idle - PWM Duty Cycle selection. Note the relationship between temperature and PWM DC can be altered by simply moving the blue dots in the curve, or by selecting the table for manual entry as shown here:



Some engines prefer additional airflow during cranking for a reliable start. This air can be automatically added only

during cranking by using the Idle - PWM Cranking Duty Cycle settings. Once the engine starts and rpm rise above the set maximum cranking rpm, the idle control switches to the previous warmup settings. Note the relationship between coolant temperature during cranking and PWM DC can be altered by simply moving the blue dots in the curve, or by selecting the table for manual entry as shown here:



NOTE: Every engine, valve type and tune is different. Suitable settings must be determined by the tuner. Do not infer any tuning settings from the images in this wiki. They are only examples.

Both 2 and 3 wire PWM idle controllers are supported. In general, the 3 wire models will provide a smoother response than the 2 wire ones, but the difference is not always significant. For 3 wire valves, 2 of the Aux outputs will be required.



Stepper Motors

Stepper motor idle controls are very common on GM and other OEM setups. These motors typically have 4 wires (bi-polar). They must be driven through power transistors or a driver module, such as the [DRV8825](#) stepper motor driver optional to the v0.4 board. These driver modules can be purchased inexpensively from a variety of vendors on sites such as eBay, Amazon, etc.

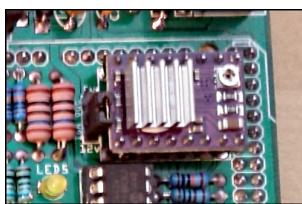
Most stepper idle valves function by turning a threaded rod in and out of the valve body in a series of partial-turn steps, increasing or decreasing airflow around the plunger (on end of valve below), and into the engine. The idle airflow bypasses the primary throttle body:



Example of a generic DRV8825 driver module on a v0.4 board:



Note the board is mounted at a standoff for air circulation and cooling:



The DRV8825 motor outputs are labeled as A2-A1-B1-B2, and the wiring connection examples are to this labeling. Check your schematics for the output connections that route to these DRV8825 outputs:

Examples of wiring to the DRV8825 driver:

The GM “screw-in” style used 1982 to 2003 on many models:

Stepper Driver Current Adjustment The DRV8825 stepper driver module includes a potentiometer (adjustable resistor) indicated by the yellow arrow in the image below. The potentiometer is used for setting the driver’s maximum current output limit. Because Speeduino uses full-step operation, the current limit is not critical to protect the module, but should be adjusted to the module’s maximum value for best operation of most automotive stepper IACs.

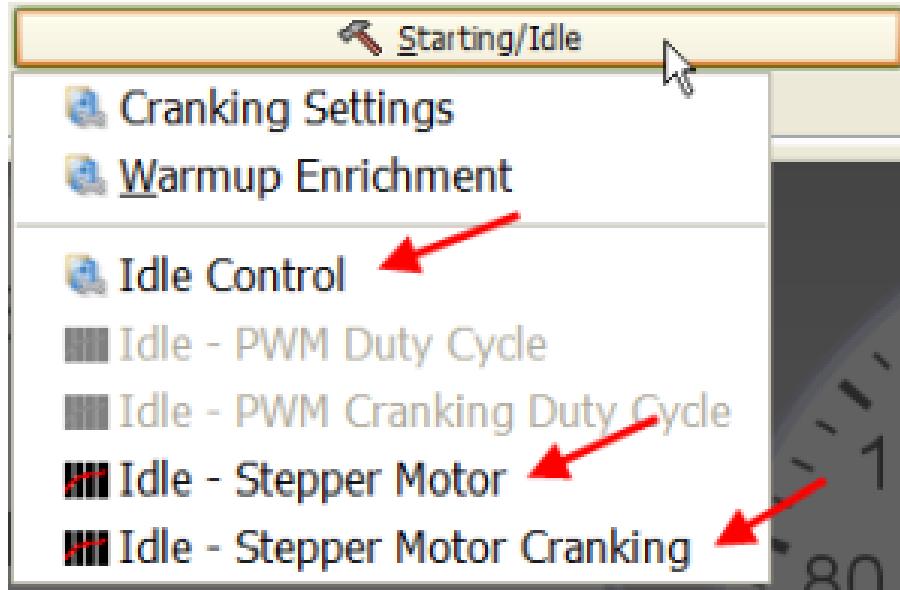


You will need a multi-meter or volt-meter to make the adjustment as outlined here. In order to set the potentiometer to maximum current before first use, ensure power to the module is OFF, then gently turn the potentiometer dial clockwise to the internal limit. **Do not force the adjustment beyond the internal stop.** Power-up Speeduino with 12V, and use the meter to test the voltage between the center of the potentiometer and any 12V ground point. Note the voltage reading. Power-down and repeat the test, this time turning the potentiometer counter/anti-clockwise gently to the internal limit. The test direction that resulted in higher voltage is the correct setting for the module.

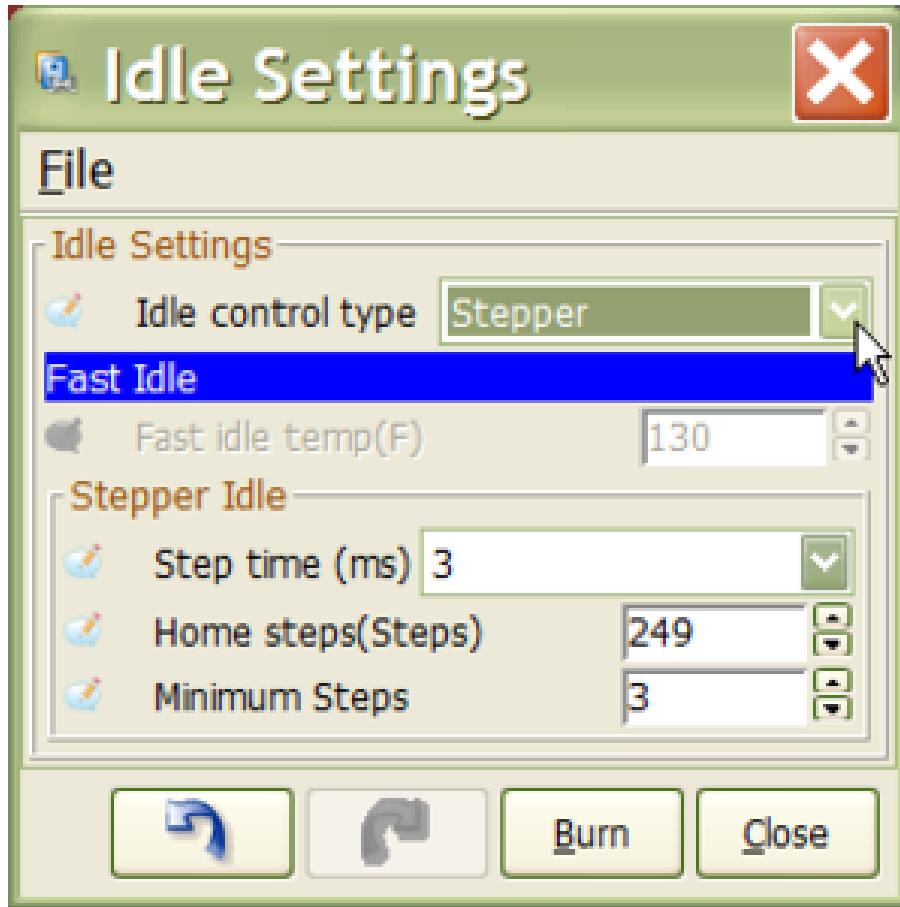
Note: Original Pololu modules are typically adjusted clockwise for maximum voltage. However, clone modules may be either clockwise or counter-clockwise, which makes this testing necessary.

The module’s rated *continuous* current is up to 1.5A. While the module can supply a peak of 2.2A of current; in full-step mode and with the potentiometer adjusted to this position, the driver is limited to approximately 70% of full current, or approximately 1.5A.

Stepper Settings Settings in TunerStudio include selecting stepper idle control, temperature and step settings for warmup, and open steps during cranking under the following selections:



Under Idle control type, stepper is selected. The basic stepper operational settings are also located in this window:

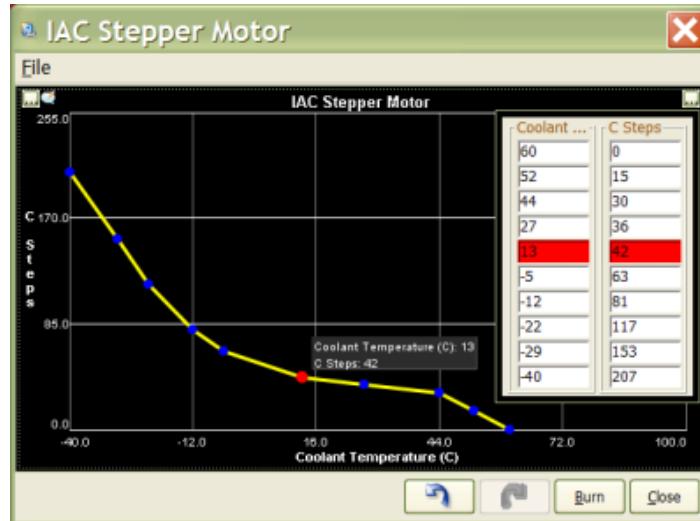


Step time: This is how long (in ms) that the motor requires to complete each step. If this is set too low, the ECU will be trying to make the next step before the previous one is completed, which leads to the motor 'twitching' and not functioning correctly. If this is set longer than needed, the system will take longer to make each adjustment, which may lead to idle fluctuating more than desired. Typical values are usually 2ms - 4ms. The common GM stepper motor requires 3ms.

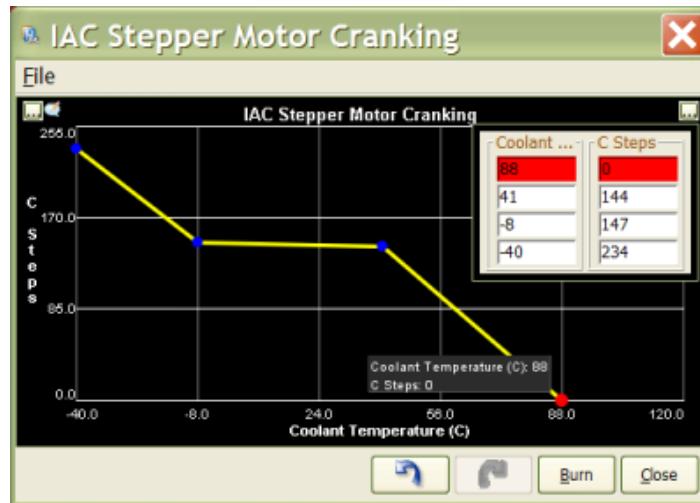
Home Steps: Stepper motors must be 'homed' before they can be used so the ECU knows their current position. You should set this to the maximum number of steps that the motor can move.

Minimum steps: In order to allow a smooth idle that isn't continually fluctuating, the ECU will only move the motor if at least this many steps are required. Typical values are in the 2-6 range, however if you have a noisy coolant signal line, this value may need to be increased.

The temperature-versus-steps is selected under the Idle - Stepper Motor selection. Note the relationship between temperature and motor steps can be altered by simply moving the blue dots in the curve, or by selecting the table for manual entry as shown here:



Some engines prefer additional airflow during cranking for a reliable start. This air can be automatically added only during cranking by using the Idle - Stepper Motor Cranking settings. Once the engine starts and rpm rise above the set maximum idle rpm, the idle control switches to the previous warmup settings. Note the relationship between coolant temperature during cranking and motor steps can be altered by simply moving the blue dots in the curve, or by selecting the table for manual entry as shown here:



NOTE: Every engine, valve type and tune is different. Suitable settings must be determined by the tuner. Do not infer any tuning settings from the images in this wiki. They are only random examples.

NOTE: Refer to the Pololu video for instructions to set the DRV8825 driver current level to maximum for most automotive full-step stepper motors.

Examples **NOTE:** While normal DSM stepper function is seen at room temperatures at 3ms, step skipping occurs just under that speed. Very cold temperatures may cause skipping, thus the recommendation of 4ms. Test for the most suitable speeds for your setup.

Chapter 4

Updating firmware

4.1 Compiling and Installing Firmware

With the goal of maximum simplicity in mind, the process of compiling and installing the firmware is reasonably straightforward.

4.1.1 Latest Stable Firmware

- **Date:** February 12th 2019
- **Details:** See <https://speeduino.com/forum/viewtopic.php?f=13&t=2599>

4.1.2 Installation - Easy Method

The simplest method of installing the Speeduino firmware onto a standard Arduino Mega 2560 is with the Speedy-Loader utility. SpeedyLoader takes care of downloading the firmware and installing it onto an Arduino without the need to manually compile any of the code yourself. You can choose the newest firmware that has been released, or select from one of the older ones if preferred. SpeedyLoader will also download the INI file for the firmware you choose so it can be loaded into your TunerStudio project.

- **Windows:** 32-bit / 64-bit
- **Mac:** SpeedyLoader-1.1.0.dmg
- **Linux:** SpeedyLoader.1.1.0.AppImage (Will need to be made executable after downloading)

Once the firmware is installed on the board, see [Connecting to TunerStudio](#) for more details on how to configure TunerStudio

4.1.3 Manually Compiling

If you want to compile the firmware yourself, or make any code changes, then the source of both the releases and the current development version is freely available. Note that manually compiling the firmware is **NOT** required to install Speeduino, the easiest (and recommended for most users) method is using SpeedyLoader as described above.

Requirements

- A Windows, Mac or linux PC
- One of the following:

- The Arduino IDE. Current minimum version required is 1.6.7, although a newer version is recommended.
- PlatformIO. Can be downloaded from <http://platformio.org/platformio-ide>
- A copy of the latest Speeduino codebase. See below.
- A copy of TunerStudio to test that the firmware has uploaded successfully

Downloading the firmware

There are two methods for obtaining the Speeduino firmware:

1. Regular, stable code drops are produced and made as releases on Github. These can be found at: <https://github.com/noisymime/speeduino/releases>
2. If you want the latest and greatest (And occasionally flakiest) code, the git repository can be cloned and updated. See <https://github.com/noisymime/speeduino>

Older firmware releases If required, older firmware releases and details can be found at [Firmware History](#)

Compiling the firmware

- Start the IDE, select *File > Open*, navigate to the location you downloaded Speeduino to and open the **speeduino.ino** file.
- Set the board type: *Tools > Board > Arduino Mega 2560* or *Mega ADK* (This is the only board currently supported)
- Click the **Verify** icon in the top left corner (Looks like a tick) 

At this point you should have a compiled firmware! If you experienced a problem during the compile, see the [Troubleshooting](#) section below.

This video walks through the whole process of installing the firmware on your Arduino from scratch:

Optional (But recommended) There is an option available for changing the compiler optimization level, which can improve . By default, the IDE uses the -Os compile option, which focuses on producing small binaries. As the size of the Speeduino code is not an issue but speed is a consideration, changing this to -O3 produces better results (Approximately 20% faster, with a 40% larger sketch size) To do this, you need to edit the platform.txt file:

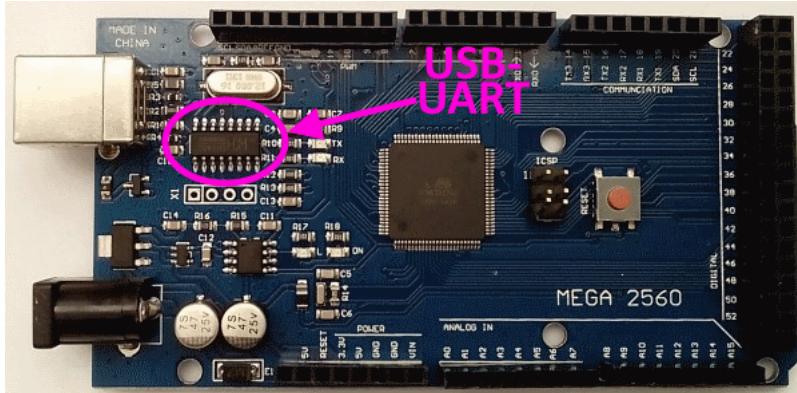
- Make sure the Arduino IDE isn't running
- Open the platform.txt file which is in the following locations:
 - On Windows: c:\Program Files\Arduino\hardware\arduino\avr
 - On Mac: /Applications/Arduino/Contents/Resources/Java/hardware/arduino/avr/
 - On Linux:
- On the following 3 entries, change the Os to be O3:
 - compiler.c.flags
 - compiler.c.elf.flags
 - compiler.cpp.flags
- Save the file and restart the Arduino IDE

Note: This is NOT required if using PlatformIO, the above optimisation is applied automatically there

Installing

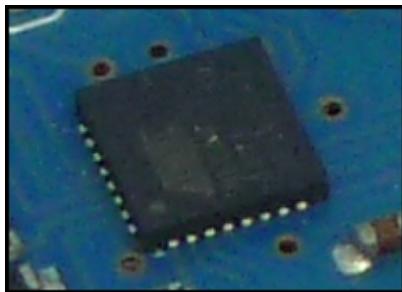
Once you've successfully compiled the firmware, installation on the board is trivial.

- Plug in your Mega 2560 to a free USB port
- If you're running an older version of **Windows** and this is the first time you've used an Arduino, you may need to install drivers for the Arduino serial chip (USB-UART or "USB adapter chip").



Most official boards and many non-official versions use the ATMega16U2 or 8U2, whereas many of the Mega2560 clone boards utilize the CH340G IC. Both types work well. The serial chips can generally be identified by appearance:

ATMega16U (square IC) - drivers included in Windows, MacOS and Linux:



or

WCH CH340G (Rectangular IC) - uses "CH341" drivers from **WCH** for Windows:



WCH-original CH340/CH341 drivers for other systems (Mac, Linux, Android, etc) may be found [here](#).

- In Arduino IDE; select the Mega2560: *Tools > Board*
- Select your system's serial port to upload: *Tools > Serial Port*
- Hit the *Upload* button from the top left corner (Looks like an arrow point to the right)



Assuming all goes well, you should see the IDE message that avrdude is done, similar to this:

```
avrduude: verifying ...
avrduude: 60612 bytes of flash verified

avrduude done. Thank you.
```

Verifying Firmware

The firmware is now loaded onto your board and you are now able to move onto [Connecting to TunerStudio](#).

Optionally, you may perform a verification of the firmware by using the Arduino IDE's Serial Monitor. This can be started by selecting 'Serial Monitor' from the Tools menu.

In the window that appears, enter a capital "S" (no quotes) and press *Enter*. The Mega should respond with the year and month of the code version installed (xxxx.xx):

```
Speeduino 2017.03
```

NOTE: Ensure the baud rate is set to 115200

You can also enter "?" for a list of queries from your Mega.

Troubleshooting

Incorrect Arduino board selected If you see the following (or similar) errors when trying to compile the firmware and the solutions:

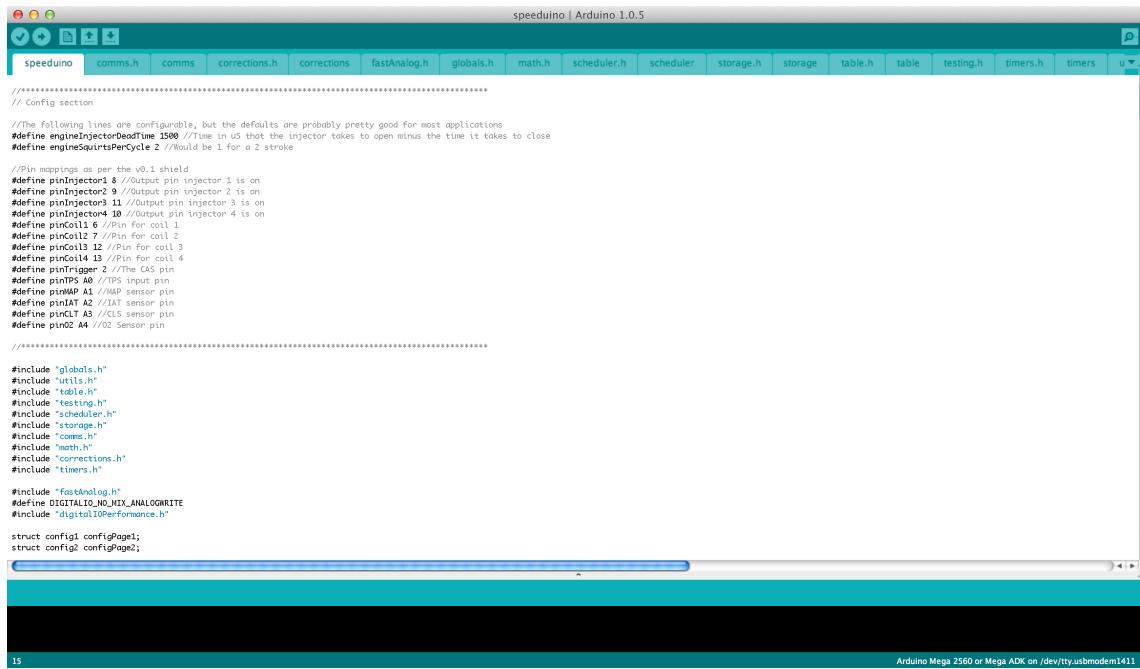
```
scheduler.ino:317:7: error: 'OCR4A' was not declared in this scope
scheduler.ino:323:8: error: 'TIMSK5' was not declared in this scope
scheduler.ino:323:25: error: 'OCIE4A' was not declared in this scope
```

You may have the wrong kind of Arduino board selected. Set the board type by selecting *Tools > Board > Arduino Mega 2560* or *Mega ADK*

Entire Speeduino project is not opened The following can occur if you have only opened the speeduino.ino file rather than the whole project.

```
speeduino.ino:27:21: fatal error: globals.h: No such file or directory
```

Make sure all the files are contained within the same directory, then select *File->Open* and find the speeduino.ino file. If you have opened the project correctly, you should have multiple tabs along the top:



The screenshot shows the Arduino IDE interface with the title bar "speedduino | Arduino 1.0.5". The code editor contains the "speedduino" sketch. The code includes comments for configuration sections and pin definitions. It also includes #include statements for various Arduino libraries like "globals.h", "math.h", "scheduler.h", etc. At the bottom of the code editor, there are two struct definitions: "config1 configPage1;" and "struct config2 configPage2;". The status bar at the bottom right indicates "Arduino Mega 2560 or Mega ADK on /dev/tty.usbmodem1411".

```
speedduino | Arduino 1.0.5

speedduino  commss.h  commms.h  corrections.h  corrections  fastAnalog.h  globals.h  math.h  scheduler.h  scheduler  storage.h  storage  table.h  table  testing.h  timers.h  timers  u.h

*****+
// Config section
//The following lines are configurable, but the defaults are probably pretty good for most applications
#define engineInjectorReadTime 1500 //Time in us that the injector takes to open minus the time it takes to close
#define engineEngineStartsPerCycle 2 //Should be 1 for a 2 stroke

//Pin mappings as per the v0.1 shield
#define pinInjector1 8 //Output pin injector 1 is on
#define pinInjector2 9 //Output pin injector 2 is on
#define pinInjector3 11 //Output pin injector 3 is on
#define pinInjector4 10 //Output pin injector 4 is on
#define pinCoil1 6 //Pin for coil 1
#define pinCoil2 7 //Pin for coil 2
#define pinCoil3 12 //Pin for coil 3
#define pinCoil4 13 //Pin for coil 4
#define pinTrigger 2 //The CAS pin
#define pinTPS A8 //TPS input pin
#define pinMAP A9 //MAP sensor pin
#define pinIAT A2 //IAT sensor pin
#define pinCLT A3 //CLS sensor pin
#define pinO2 A4 //O2 Sensor pin

*****+
#include "globals.h"
#include "utils.h"
#include "math.h"
#include "timing.h"
#include "scheduler.h"
#include "storage.h"
#include "commss.h"
#include "math.h"
#include "corrections.h"
#include "timers.h"

#include "fastAnalog.h"
#define DIGITALIO_NO_MHZ_ANALOGWRITE
#include "digitalIOPerformance.h"

struct config1 configPage1;
struct config2 configPage2;

15
Arduino Mega 2560 or Mega ADK on /dev/tty.usbmodem1411
```

[\[Coll-attribution-page\]](#)