I. Introduction ........................................................................................................................................... 1
   A. Hardware features ............................................................................................................................ 1
   B. Software features ........................................................................................................................... 1
II. Software ................................................................................................................................................ 3
    A. Installation ...................................................................................................................................... 3
    B. Versions ......................................................................................................................................... 3
III. Connecting display software to IntelliJect ......................................................................................... 4
     A. Connection methods ..................................................................................................................... 4
     B. Connection Status ....................................................................................................................... 5
IV. Menus .................................................................................................................................................. 6
    A. File menu ....................................................................................................................................... 6
    B. Comm menu ................................................................................................................................... 6
    C. Windows menu .............................................................................................................................. 7
    D. Toolbars menu .............................................................................................................................. 8
    E. Help menu ..................................................................................................................................... 8
V. IntelliJect Configuration ....................................................................................................................... 10
    A. Changing configuration data ......................................................................................................... 10
    B. Tables .......................................................................................................................................... 11
    C. Throttle ....................................................................................................................................... 19
    D. Governor ..................................................................................................................................... 24
    E. Engine .......................................................................................................................................... 29
    F. Cooling ......................................................................................................................................... 34
    G. Injectors ....................................................................................................................................... 36
    H. Pump ............................................................................................................................................ 36
    I. Sensors .......................................................................................................................................... 41
    J. Comms .......................................................................................................................................... 47
    K. Locking ......................................................................................................................................... 51
    L. Maintenance Schedule .................................................................................................................. 55
    M. Configuration files ....................................................................................................................... 57
VI. How To Configure And Calibrate IntelliJect .................................................................................... 60
    A. Hardware setup ............................................................................................................................. 60
    B. Calibration .................................................................................................................................... 63
    C. Polishing the calibration ............................................................................................................... 65
VII. Engine Logbook ................................................................................................................................. 67
XXVII. Test Mode ........................................................................................................... 99
   A. Output tests ........................................................................................................... 99
   B. Crank error test .................................................................................................... 100
   C. Calibrate servos against hard stop ....................................................................... 101
   D. Internal tests ......................................................................................................... 101
   E. Misfire tests .......................................................................................................... 102

XXIII. Acceptance Test Report ................................................................................... 103
   A. User provided data ............................................................................................... 104
   B. Acceptance test scores ....................................................................................... 105

XXIV. Simulator ............................................................................................................ 107

XXV. Firmware update .................................................................................................. 108
   A. Firmware distribution ........................................................................................... 108
   B. Programming the firmware ................................................................................ 109
   C. Bootloader communications over USB ............................................................... 110
   D. Bootloader communications over UART ........................................................... 110
   E. Bootloader communications over CAN ............................................................... 110

XXVI. Log files .............................................................................................................. 111
   A. Logging toolbar .................................................................................................... 111
   B. Replaying an efitel file. ...................................................................................... 111

XXVII. External devices .................................................................................................. 113
   A. Cooling Fan .......................................................................................................... 113
   B. Horiba Gas .......................................................................................................... 115
   C. Sound .................................................................................................................. 115
   D. Fuel Pump / Meter .............................................................................................. 116
   E. NI Data Acquisition ............................................................................................ 117

XXVIII. Hardware ........................................................................................................... 118
   A. Enclosure Dimensions ....................................................................................... 118
   B. Electrical Connections ....................................................................................... 119

XXIX. Software Release Notes ...................................................................................... 124
   A. Version 1.8.939 (6 December 2019) ................................................................. 124
   B. Version 1.7.806 (3 December 2019) ................................................................. 127
   C. Version 1.7.805 (7 November 2019) ................................................................. 128
D. Version 1.7.803 (31 October 2019) ................................................................. 128
E. Version 1.7.800 (13 October 2019) ................................................................. 129
F. Version 1.7.799 (18 September 2019) ......................................................... 130
G. Version 1.7.798 (13 August 2019) ................................................................. 130
H. Version 1.7.797 (1 August 2019) ................................................................. 130
I. Version 1.7.796 (19 July 2019) ................................................................. 131
J. Version 1.7.795 (9 July 2019) ................................................................. 131
K. Version 1.6.742 (8 November 2019) ........................................... 133
L. Version 1.6.741 (1 August 2019) ................................................................. 134
M. Version 1.6.738 (9 July 2019) ................................................................. 134
N. Version 1.6.737 (19 June 2019) ................................................................. 134
O. Version 1.6.736 (11 June 2019) ................................................................. 135
P. Version 1.6.735 (21 May 2019) ................................................................. 135
Q. Version 1.6.733 (25 April 2019) ................................................................. 135
R. Version 1.6.732 (3 April 2019) ................................................................. 135
S. Version 1.6.729 (7 March 2019) ................................................................. 136
T. Version 1.6.725 (26 February 2019) ................................................................. 136
U. Version 1.5 (1 February 2019) ................................................................. 138
V. Version 1.4.607 Update 1 (25 October 2018) ........................................... 140
W. Version 1.4.606 (18 October 2018) ......................................................... 141
X. Version 1.3 (11 August 2018) ................................................................. 144
Y. Version 1.2 (4 April 2018) ................................................................. 148
Z. Version 1.1 (5 February 2018) ................................................................. 150
XXX. Glossary ..................................................................................... 152
FIGURE 1: INTELLiJECT SOFTWARE INSTALLATION ON WINDOWS 10 ..................................................................................3
FIGURE 2: CONNECTION DIALOG, USED TO CONFIGURE HOW THE DISPLAY SOFTWARE CONNECTS TO INTELLiJECT. .................................................................4
FIGURE 3: INTELLiJECT CONNECTION STATUS, IN THE STATUS BAR ..................................................................................5
FIGURE 4: FILE MENU ........................................................................................................................................................6
FIGURE 5: COMM MENU ....................................................................................................................................................6
FIGURE 6: WINDOWS MENU ................................................................................................................................................7
FIGURE 7: TOOLBARS ......................................................................................................................................................8
FIGURE 8: ABOUT MENU ..................................................................................................................................................8
FIGURE 9: ABOUT INTELLiJECT DISPLAY ........................................................................................................................9
FIGURE 10: PROMPT FOR DOWNLOAD A NEW VERSION OR A BUG FIX UPDATE ........................................................................9
FIGURE 11: GLOBAL FEATURES OF THE CONFIGURATION WINDOW, WITH THE CONFIGURATION LOCKED .................10
FIGURE 12: ANNOTATED FUEL TABLE ..........................................................................................................................11
FIGURE 13: TABLE LIST ..................................................................................................................................................12
FIGURE 14: TABLE DISPLAY SHOWING TWO TABLES AT ONCE .........................................................................................13
FIGURE 15: THROTTLE CONFIGURATION .......................................................................................................................19
FIGURE 16: STRIP CHART SHOWING RESULTS OF AN ENGINE CHECKOUT, DEFINING THE TIME RANGE FOR THE THROTTLE CURVE BUILD ........................................21
FIGURE 17: WARNING DIALOG INDICATING INSUFFICIENT DATA TO BUILD A THROTTLE CURVE .........................22
FIGURE 18: THE CURVE BUILDER WINDOW SHOWING INPUT DATA AND A COMPUTED THROTTLE CURVE ...................22
FIGURE 19: THE CURVE BUILDER WINDOW SHOWING A USER SPECIFIED EXPECTED RPM CURVE ...............................23
FIGURE 20: RPM GOVERNOR SETTINGS, AND CHART OF RPM CONTROL PERFORMANCE UNDER DIRECT COMMAND .......24
FIGURE 21: RPM GOVERNOR DRIVEN BY THROTTLE COMMAND, 10% TO 80% AND BACK ........................................25
FIGURE 22: RPM LIMITER, WITHOUT AND WITH THROTTLE RATE LIMITING ..............................................................27
FIGURE 23: ENGINE CONFIGURATION ........................................................................................................................29
FIGURE 24: ANNOTATED DIAGRAM OF A DIRECTIONAL CRANK WHEEL WITH A COUNT OF 20 ........................................31
FIGURE 25: COOLING CONFIGURATION .........................................................................................................................34
FIGURE 26: INJECTORS CONFIGURATION AND TELEMETRY ...........................................................................................36
FIGURE 27: PUMP CONFIGURATION ............................................................................................................................41
FIGURE 28: PUMP DUTY CYCLE AND FUEL PRESSURE WITH THE BANG-BANG AND CONTINUOUS CONTROLLER AT CONSTANT THROTTLE ........................................42
FIGURE 29: SENSOR CONFIGURATION WINDOW, AND THE SENSOR SELECTION DROP DOWN ........................................43
FIGURE 30: AN EXAMPLE OF FAILED SENSOR TELEMETRY ..........................................................................................46
FIGURE 31: COMMUNICATIONS CONFIGURATION ........................................................................................................47
FIGURE 32: LOCKING WINDOW, WITH THE CONFIGURATION UNLOCKED .............................................................................51
FIGURE 33: CONFIGURATION LOCKED, ENTER UNLOCK PASSWORD IS VISIBLE ...............................................................51
FIGURE 34: NOT-ACKNOWLEDGE MESSAGE ..................................................................................................................52
FIGURE 35: INTELLiJECT IN PARTIAL UNLOCK STATE .......................................................................................................52
FIGURE 36: NOT-ACKNOWLEDGE MESSAGE DUE TO RUNNING ENGINE ...........................................................................54
FIGURE 37: MAINTENANCE SCHEDULE CONFIGURATION .................................................................................................55
FIGURE 38: CONFIGURATION SELECTION SAVE WINDOW ................................................................................................57
FIGURE 39: CONFIGURATION SELECTION WINDOW FOR OPENING CONFIGURATION DATA .............................................58
FIGURE 40: CONFIGURATION COMPARISON WINDOW .....................................................................................................59
FIGURE 41: CONFIGURATION COMPARISON REPORT .........................................................................................................59
FIGURE 42: INJECTOR TEST EXAMPLE .............................................................................................................................61
FIGURE 43: LOGBOOK WINDOW WITH CONTEXT MENU OPEN ..........................................................................................67
FIGURE 44: LOGBOOK ENTRY DIALOG ...........................................................................................................................67
FIGURE 45: ENGINE WEAR AND MAINTENANCE STATUS ................................................................................................69
FIGURE 46: ENGINE WEAR INPUT .....................................................................................................................................69
FIGURE 47: MAINTENANCE ENTRY DIALOG ........................................................................................................................70
FIGURE 48: INTELLiJECT STICKY AND REGULAR ERROR WINDOWS WITH AND WITHOUT ERRORS ...............................75
FIGURE 49: ENABLE TOOLBAR WITH AN ERROR VISIBLE ..................................................................................................75
FIGURE 50: SYSTEM WINDOW ........................................................................................................................................77
FIGURE 51: EXCEPTION ....................................................................................................................................................78
FIGURE 52: RESET DATA ..................................................................................................................................................78
FIGURE 53: SEQUENCE OF STEPS TO SETUP AN SD CARD ...............................................................................................79
www.power4flight.com

FIGURE 54: THE FILES THAT RESULT FROM SETTING UP A CARD ................................................................. 80
FIGURE 55: SD CARD STATUS WHEN THE CARD IS CONFIGURED AND RECORDING ..................................................... 81
FIGURE 56: SELECTING SD CARD DIRECTORY TO PARSE..................................................................................... 82
FIGURE 57: SELECTING SD RECORD......................................................................................................................... 82
FIGURE 58: SD PARSE OUTPUTS .................................................................................................................................. 82
FIGURE 59: SELECTING AN SD RECORD FOR TRANSFER ............................................................................................. 83
FIGURE 60: SD TRANSFER FINISHED ......................................................................................................................... 83
FIGURE 61: DEFAULT DISPLAY LAYOUT ...................................................................................................................... 84
FIGURE 62: GAUGES ....................................................................................................................................................... 85
FIGURE 63: VARIABLES .................................................................................................................................................. 85
FIGURE 64: EXAMPLE STRIP CHART ........................................................................................................................... 86
FIGURE 65: CHART CONFIGURATION WINDOW AND SIGNAL SELECTION ................................................................. 86
FIGURE 66: ENABLE TOOLBAR WITH ALL ENABLED AND WITH ALL DISABLED, SPARK 3 ENABLED............................ 89
FIGURE 67: THROTTLE AND RPM TOOLBARS ....................................................................................................... 89
FIGURE 68: PROFILE RUNNER TABLE AND CHART TABS, SHOWING A RUNNING PROFILE WITH MIXED THROTTLE AND RPM ................................................................. 90
FIGURE 69: PROFILE BUILDER WINDOW ................................................................................................................ 91
FIGURE 70: ALARM DISPLAY WITH AND WITHOUT ALARMS ACTIVE .................................................................. 92
FIGURE 71: ALARM TOOLBAR WITH NO ALARMS, ALARMS ACTIVE, AND ALARMS ACKNOWLEDGED .............. 92
FIGURE 72: PACKET LOG WINDOW ........................................................................................................................ 93
FIGURE 73: PACKET LOG WINDOW SHOWING TELEMETRY WITH AND WITHOUT SUMMARY PACKETS .................. 93
FIGURE 74: TABLE VISUALIZATION SHOWING ALL VIEWS OF A FUEL TABLE ........................................................... 94
FIGURE 75: VISUALIZATION OF A TABLE SUBSET .................................................................................................. 95
FIGURE 76: OSCILLOSCOPE WINDOW SHOWING A SCOPE CAPTURE ...................................................................... 96
FIGURE 77: SCOPE SIGNALS WINDOW AND LIST OF CHANNELS ............................................................................ 97
FIGURE 78: TEST MODE WINDOW WITH TEST NOT RUNNING AND RUNNING ......................................................... 99
FIGURE 79: ACCEPTANCE TEST REPORT WINDOW BEFORE BEING FILLED OUT ..................................................... 103
FIGURE 80: ACCEPTANCE TEST REPORT AFTER PUTTING IN REQUIRED DATA ....................................................... 104
FIGURE 81: ACCEPTANCE SCORES ............................................................................................................................. 105
FIGURE 82: SIMULATOR DISPLAYS ............................................................................................................................. 107
FIGURE 83: VERSION MISMATCH QUERY .................................................................................................................. 108
FIGURE 84: PROGRAMMING TOOL, CONNECTED TO INTELLIJECT .............................................................................. 108
FIGURE 85: PROGRAMMING IN PROGRESS, CONNECTED TO BOOTLOADER ............................................................. 109
FIGURE 86: INTELLIJECT IN BOOTLOADER QUESTION ............................................................................................ 109
FIGURE 87: LOGGING TOOLBAR ................................................................................................................................. 111
FIGURE 88: REPLAY TOOLBAR ................................................................................................................................... 112
FIGURE 89: DISCONNECTED AND CONNECTED COMMAND WINDOWS FOR THE COOLING FANS ......................... 113
FIGURE 90: FAN CONTROL AND INTERFACE TABS .................................................................................................. 114
FIGURE 91: COOLING FAN TOOLBAR ......................................................................................................................... 114
FIGURE 92: LEFT HORIBA IN STANDBY, RIGHT HORIBA MEASURING ........................................................................ 115
FIGURE 93: HORIBA TOOLBAR, CONNECTED AND DISCONNECTED ......................................................................... 115
FIGURE 94: SOUND METER ........................................................................................................................................ 115
FIGURE 95: SOUND METER TOOLBAR, CONNECTED AND DISCONNECTED ............................................................... 116
FIGURE 96: TRIPLEX AND MAX MACHINERY ........................................................................................................... 116
FIGURE 97: FUEL METER TOOLBAR WITH MAX AND TRIPLEX .............................................................................. 117
FIGURE 98: NI WINDOW .............................................................................................................................................. 117
FIGURE 99: INTELLIJECT ENCLOSURE DIMENSIONS ................................................................................................. 118
FIGURE 100: ENGINE CONNECTOR DEVELOPMENT HARNESS: NOT RECOMMENDED FOR FLIGHT .................... 120
FIGURE 101: VEHICLE CONNECTOR DEVELOPMENT HARNESS: NOT RECOMMENDED FOR FLIGHT .................... 122
I. INTRODUCTION

IntelliJect is an electronic fuel injection system from Power4Flight designed specifically for use in small engine aerospace applications.

A. Hardware features

- Onboard power conditioning (8-30 Volts in)
- Dual injector outputs with fault detection
- Dual CDI ignition output
- Fuel pump control output
- PWM outputs for throttle and cowl flap control
- Manifold pressure sensor
- Dual redundant barometric pressure sensors
- Dual redundant crank sense inputs
- Dual redundant CHT sensor inputs
- Manifold air temperature input
- Spare temperature input
- Fuel pressure input
- Analog and PWM throttle position or command input
- CAN, USB, and Serial communications
- Onboard SD card data logging
- Compatible with Currawong EFI accessories
- Weights
  - Board Only: 29.0g (1.02oz)
  - With Enclosure: 71.3g (2.51oz)
- Size
  - Board Only: 45mm x 75mm (1.77” x 2.95”)
  - With Enclosure: 48mm x 79 (1.89” x 3.11”)

B. Software features

- Configurable for a variety of engine types (two-stroke, four-stroke, triples\(^1\), twins and singles)
- Seamless integration with third party systems, particularly flight controllers
- Free communications ICD and software developers kit
- Multiple communication protocols supported
- Sophisticated and robust firmware designed for high reliability aerospace applications
- Choice of alpha-n or speed-density fuel injection with multi-variable compensation
- Direct throttle command, RPM control, or throttle sensing
- Configurable throttle limiting based on temperature, speed, and altitude.

\(^1\) IntelliJect rev3 supports three injectors and ignitions, dual cowl flaps, and expanded power input options.
• Spark and injection interruption for rev limiting
• Injector skipping to improve injector dynamic range
• Closed loop cooling control using CHT sensors
• Fuel pump control using proportional or bang-bang fuel pressure feedback
• Fault detection and correction for Baro, MAT, MAP, CHT, fuel pressure, and crank sense
• Onboard electronic log-booking and maintenance tracking

This is the user’s manual for the IntellJect system. All features of the system have a corresponding interface in the display software; accordingly, this manual is organized around the interface software.
II. SOFTWARE

All the software, firmware, documentation, and the standard developers kit (SDK) are contained within the PC software. The software is available as a windows installer, a macOS app bundle, or a Linux zipped archive. The windows installer is available from the Power4Flight website.

A. Installation

The software requires:

- A personal computer with Windows 7 (or later) -or- macOS 10.10.5 (or later) -or- Linux².
- A USB or RS-232 port.

The Windows installer is conventional, with options to place shortcuts on the user’s desktop and start menu. It will also install an un-installer to remove the software. On macOS and Linux the software is uninstalled by deleting the app bundle or directory.

The software (all operating systems) will store data in the user’s directory: “~/Power4Flight/IntelliJect Display”. The uninstaller will not remove this directory, but it can be deleted by the user if desired.

B. Versions

IntelliJect software uses a three-part versioning scheme; for example, 1.7.796 is major version 1, minor version 7, and build number 796. Changes to the major version number are very rare and indicate large system wide architectural changes to IntelliJect. Forwards or backwards compatibility between major versions is not guaranteed, or even likely. The minor version will change anytime IntelliJect receives a new feature. Minor version changes typically maintain compatibility so that upgrading (and in most cases downgrading) the minor version will not invalidate the configuration.

The build number will change anytime new software is released. When a bug fix is released the build number will change but the major and minor version will not. The build number does not reset to zero when the minor or major numbers are changed; so, for example, the first release of version 1.7 was build 795, and the second release of version 1.7 was build 796. Release notes are available in section XXIX of this manual, or from the software (section IV.E).

---

² This manual uses screenshots from the software running on Windows 7 and 10. Your version may look different if you use a different operating system. The Linux version is built and tested on Mint Linux.
III. CONNECTING DISPLAY SOFTWARE TO INTELLIJECT

IntelliJect has a UART (Serial RS-232) interface, CAN interface, and USB interface\(^3\) which can be used to connect it to the display software. Note that the USB port will provide power to the IntelliJect main processor, but the injectors, ignitions, servos, and sensors are not powered by the USB. If you want to work with those systems, you must supply main power to IntelliJect. It is not recommended to use the USB interface with a running engine.

After connecting IntelliJect to the computer and applying power, use the **Comm->Connect** menu to bring up the connection dialog. This dialog is used to configure how the display software connects to IntelliJect. The current connection method is always visible in the status bar.

![Connection dialog](image)

**Figure 2:** Connection dialog, used to configure how the display software connects to IntelliJect.

A. Connection methods

1. Serial

For RS-232 or USB connection to IntelliJect you use the serial option. The RS-232 interface defaults to 57600 bits per second. In the case of the USB connection the baud rate does not matter. Note that the USB interface will appear as an “STMicroelectronics Virtual COM Port”. Each time the connection dialog is displayed software rescans the available ports, so if you plug in a USB serial device after opening this dialog you will need to close and re-open it. The first time you plug in the IntelliJect USB port you may need to wait while Windows downloads the necessary driver for the STMicroelectronics virtual com port.

\(^3\) The USB interface is currently disabled in software (version 1.7 and earlier). The USB interface will be enabled in later firmware versions.
2. CAN

For the controller area network (CAN) option you will need a PC peripheral that implements CAN. On windows we typically use the Systec USB-CAN module, however there is software support for other CAN hardware (see section XXVIII.B.8). The CAN interface runs at 1Mbit per second by default, but it can be configured for different speeds. In general, the CAN settings in the connection dialog should be complementary to the IntelliJect communications settings (see section V.J, or the communications ICD, for more information).

3. Network client

The network client option allows the display software to connect to another instance of the display software which is relaying IntelliJect communications over the local network. To use this option you must supply the IP address and port number of the TCP server which is acting as the data relay.

4. Network server

The network server option allows the display software to act as a TCP server relaying IntelliJect data. This is the only option which can be simultaneously selected with other communication options. When using this option, you must select the port number that the server will listen on for client connections. You will need to allow IntelliJect Display to open the port in your operating systems firewall.

5. Simulator

The simulator option starts a very simple engine and IntelliJect simulator. The simulator runs the same software as the IntelliJect hardware; and includes a (very) rudimentary engine model. The simulator can be used to learn about the IntelliJect system and its software.

B. Connection Status

The connection status is always visible in the lower right of the application on the status bar, with a green light to indicate the connection is online or offline. A connection goes offline if no data is received from IntelliJect for 2 seconds.

![Figure 3: IntelliJect connection status, in the status bar.](image)

When the display software connects to IntelliJect it will request the configuration information needed to populate and configure the displays.
IV. MENUS

A. File menu

- **Open IntelliJect Config...** asks the user to choose a file that contains IntelliJect configuration data, see section V.L for more information. The software will send the configuration data to IntelliJect if it is online. Otherwise the display will simply be populated with the data loaded from the file. If this option is greyed out it is because IntelliJect is online but locked, see section V.K
- **Save IntelliJect Config...** asks the user to choose a file for saving IntelliJect configuration data. The user will be prompted to specify which data should be saved to the file. This file can later be opened with the Open IntelliJect Config... menu option.

- **Compare IntelliJect Config...** opens a tool for comparing IntelliJect configurations, see section V.M.3 for more information.
- **Acceptance Test Report...** opens a tool for generating an engine checkout acceptance test report, see section XXIII for more information.
- **Explore log files** will open the system’s file explorer (windows explorer or macOS finder) to the directory that contains the log files (~/Power4Flight/IntelliJect Display/).
- **Reset log file** will cause the current log file to be closed and a new file started.
- **Open replay file...** asks the user to supply a replay file (extension “.efitel”) which is used for data replay. See section XXVI.B for more information.
- **SD Card Parse...** asks the user to select SD card files to parse, converting to normal log files. See section XII for more information.

B. Comm menu

- **Connect...** opens the IntelliJect connection dialog. See section III for more details.
- **Disconnect** shuts down all IntelliJect connections, including the simulator.
- **Send All** sends all configuration data to the IntelliJect. If this option is greyed out it is because IntelliJect is offline or locked, see section V.K
- **Request All** requests all configuration data from the IntelliJect.

- **Undo** sends a packet to undo the previous configuration change command. If this option is greyed out it is because IntelliJect is locked, or no undo packets are in the undo buffer. The number in parenthesis (3 in Figure 5) gives the number of undo commands in the buffer.
**Redo** sends a packet to redo the last undo operation. If this option is greyed out it is because IntelliJect is locked, or no redo packets are in the redo buffer. The number in parenthesis (1 in Figure 5) gives the number of redo commands available.

C. **Windows menu**

- **Open display layout**... asks the user to supply a “.ini” file that specifies the layout of the windows (Section XIII).
- **Save display layout**... asks the user to choose a filename to save the current display layout to a file.
- **Default display layout** restores the windows to default layout.
- **Lock display layout** toggles the lock feature which prevents the display layout and size from being changed.
- **Fullscreen** toggles between full screen and normal mode.
- **Chart #** toggles the display of a chart on or off. The number of charts is configurable (Chart 0, Chart 1, etc.) (Section XV).
- **Open display layout...** asks the user to supply a “.ini” file that specifies the layout of the windows (Section XIII).
- **Save display layout...** asks the user to choose a filename to save the current display layout to a file.
- **Default display layout** restores the windows to default layout.
- **Lock display layout** toggles the lock feature which prevents the display layout and size from being changed.
- **Fullscreen** toggles between full screen and normal mode.
- **Chart #** toggles the display of a chart on or off. The number of charts is configurable (Chart 0, Chart 1, etc.) (Section XV).
- **Add Chart** creates a new chart.
- **Remove Chart...** asks the user to choose a chart to remove.
- **Configuration** toggles the configuration window (Section IV).
- **2nd Configuration** toggles a clone of the configuration window.
- **Gauges** toggles the gauges window (Section XVII).
- **Errors** toggles the errors window (Section VI).
- **Sticky Errors** toggles the sticky errors (Section VI).
- **Alarms** toggles the alarms window (Section XVIII).
- **System** toggles the system window (Section XI).
- **Profile Runner** toggles the profile runner window (Section XX).
- **Maintenance** toggles the maintenance status (Section VIII).
- **Logbook** toggles the engine logbook (Section VI).
- **Crank Timing** toggles the crank timing window.
- **Oscilloscope** toggles the oscilloscope window (Section XX).
- **Packet Log** toggles the packet log window (Section XIX).
- **Table Visualization** toggles the visualization (Section XX).
- **Test Mode** toggles the test mode window (Section XXII).
- **Simulator** toggles the simulator window (Section XXIV).
- **Cooling Fan** toggles the cooling fan window (Section XXIII).
- **Horiba Gas** toggles the Horiba gas window (Section XXVII.B).
- **Sound** toggles the sound meter window (Section XXVII.C).
- **Fuel Pump/Meter** toggles the fuel pump meter (Section XXVII.D).
- **NI Data Acquisition** toggles the National Instruments data acquisition window (section XXVII.E).
D. Toolbars menu

- **EFI enable** toggles the enable toolbar (section XVI.A).
- **Logging** toggles the logging toolbar (section XXVI.A).
- **Throttle** toggles the throttle command toolbar (section XVI.B).
- **RPM** toggles the rpm command toolbar (section XVI.B).
- **Alarms** toggles the alarm toolbar (section XVIII).
- **Fan** toggles the cooling fan toolbar (section XXVII.A.4).
- **Horiba** toggles the gas analyzer toolbar (section XXVII.B.1).
- **Sound** toggles the sound meter toolbar (section XXVII.C).
- **Fuel Pump/Meter** toggles the fuel meter toolbar (section XXVII.D).
- **Replay** toggles the replay toolbar (section XXVI.B).

![Figure 7: Toolbars.](image)

E. Help menu

- **Users Guide** copies the user’s guide (this document) to the user’s directory (~/Power4Flight/IntelliJect Display/Docs) and launches the system viewer to display it.
- **Communications Reference** copies the communications ICD to the user’s directory (~/Power4Flight/IntelliJect Display/Docs) and launches the system viewer to display it.

![Figure 8: About menu.](image)

- **Export communications SDK...** asks for a location to write the communications standard developers kit. The kit is used to build software that interfaces with, or works with, IntelliJect.
- **About IntelliJect Display...** shows the version of the display software, see Figure 9 below.
- **Check for Updates...** checks the Power4Flight web site to determine if a newer software version is available, see Figure 10 below.
- **Firmware Update...** invokes the window used to update the firmware on IntelliJect. See section XXV for details.
- **Open Source Software** provides details of the open source software the software uses.
The version information in Figure 9 is for the software on the PC, not the firmware on Intelliject. The software has a major and minor version number (1.8 in Figure 9), a “Testing” or “Release” indicator, and a build number (855 (0x08AEE842) in Figure 9). The **Show Release Notes...** button will display the software release notes in your browser.

If you are connected to the internet the About window will query the Power4Flight website to determine what versions of software are available. In Figure 9 the latest version of software is 1.7.796, and you can click on the link to go directly to that download.

If a software update is available you will be prompted to download it. Select **Do not ask again for these versions** to suppress future prompts. In Figure 10 the version prompts were created by an older (1.6.720) version of software. The first prompt is indicating that a newer version (1.7) is available; and the second prompt is indicating that a bug fix build (1.6.738) is available for the version currently running. If you suppress the update prompt use the menu item **Check for Updates...** to override.
V. **INTELLJECT CONFIGURATION**

IntelliJect is configured in categories; each category is configured with a packet from the communications ICD. The configuration window has a tab for each category; and the top of the window has controls common to all the configuration categories.

A. Changing configuration data

Each time IntelliJect boots up the configuration is locked and cannot be changed without first unlocking. This is a safety mechanism to prevent unintended changes. To change configuration, you first unlock the configuration using the **Unlock** button at the top of the configuration window, see section V.K for more information. You do not need to unlock to run an engine.

The **Send** button is used to send the configuration data of the active tab to IntelliJect. The **Request** button is used to request the configuration data of the active tab from IntelliJect. Typically, you will not use these buttons, as editing the data in the tab will trigger an immediate send of the data, and IntelliJect will immediately respond with the updated data. The **Open** button reads configuration data from a file, for the active tab, and sends it to IntelliJect.

![Configuration Window](image)

Figure 11: Global features of the configuration window, with the configuration locked.

When sending or requesting data the buttons will turn red while waiting for IntelliJect to respond. Seeing the button turn red and then back to grey is confirmation that the displayed configuration data are up to date. Any time you send configuration data (if IntelliJect is unlocked) the data are immediately written to nonvolatile storage; and will be automatically reloaded the next time IntelliJect boots up, see section V.K.3 for more information.

If you send a configuration change packet to IntelliJect the display software will buffer an undo packet. The undo packet is the packet that returns the configuration to the state it was prior to sending the configuration change. You can undo a configuration change by using the keyboard shortcut ctrl-z, or by selecting the **Undo** command from the **Comm** menu (section IV.B). If you perform an undo operation a redo packet will be buffered by the software. A redo is simply an undo of an undo. You can redo a change by using the keyboard shortcut ctrl-y, or by selecting the **Redo** command from the **Comm** menu. Undo/redo is not a feature of IntelliJect itself, it is entirely within the display software. Restarting the display software will clear the buffers.
B. Tables

Much of the engine calibration is performed using tables. All tables have two dimensions with a variable number of rows or columns, up to a maximum of 21 rows and 15 columns. A table consists of the following data:

- The number of rows and columns (the number can be zero – which disables the table).
- Indices for each row and each column.
- A value for each cell of the table.
- A color for each cell of the table.
- A global enable flag.
- A flag to select if the rows are throttle or %MAP/Baro. This flag only applies to some tables.

The interpretation of the table indices and values are specific to the table, see below for the details of each table. IntelliJect will perform bilinear interpolated lookups into the table to determine its output. Note that lookups will not extrapolate beyond the range of the table indices. If a table is disabled, or if it has zero rows or zero columns, the table output will be 0.

In Figure 12 you can see a table annotated to show key features. The table in Figure 12 is enabled and has 12 Throttle rows and 8 RPM columns. The engine is currently operating at a throttle of 22.0% and 4100 RPM, as evidenced by the red box. The output from this table 24.76 (μg/cc/rev), as given at the top of the table.

Figure 12: Annotated fuel table
To insert a row or column in the table use the **Ins.** button on the left or top of the table respectively. The software will ask for the index value of the new row or column, and then perform the insert, choosing cell values by interpolating from the current cells and indices. To delete a row or column use the **Del.** Buttons. You can select one or more rows or columns to delete, or if you select nothing, the software will prompt you for the index of the row or column to delete. Use the **Clear** button to remove all rows and columns from a table.

To change the value of a cell select it and enter numbers from the keyboard. Alternately you can double click on the cell (or select and press enter) and the software will ask for the new value. To change the value of a row or column index: double click the index and answer the prompt. To change the value of more than one cell at a time you can select multiple cells or multiple rows and columns, and then use the **Set, Add** or **Mult** buttons. The **Set** button prompts for a new value and sets all the selected cells to that value. The **Add** button does the same; but sums the input with previous cell values. The **Mult** button works by multiplying the input against the previous cell values.

The interpolate buttons (**Interp, Row Itrp., Col Itrp.**) set the value of a cell by interpolating surrounding cells. You must select three or more cells in a row or column; the value of the inner cells of the selection will be set according to a linear interpolation from the surrounding cells. **Row Itrp.** interpolates across rows if you have 3 or more rows selected. **Col Itrp.** interpolates across columns if you have 3 or more columns selected. **Interp** interpolates across both rows and columns if you have 3 or more rows and columns selected.

Notice the difference between inserting a row or column; and changing the value of the row or column index. Using insert, the new row or column will have values which are interpolated to match the new index. However, changing the index value does not change the cell values.

The **MAP** checkbox is used to select if the table row axis is indexed by load (percent ratio of manifold to barometric pressure) or by throttle. The former method is referred to as speed-density, and the latter as alpha-n. The **Convert** button can be used to switch from alpha-n to speed-density and vice versa. To do this the table must be one in which the rows are index by throttle/load and the columns are indexed by RPM. The conversion will only be performed if the MAP Estimate table is populated with data.

The table enable feature is useful for turning a table on or off without discarding the information in the table. Although you would never turn a fuel table off, there are other tables which you may want to be able to turn on and off as you develop the configuration.
Cells of the table can be colored. The color information is not used by Intelliject; but it allows you to indicate meta-information about the cell. For example, coloring a cell yellow might indicate that it needs more adjustment, or green may indicate best economy. Select cells and use the **Color** button to set the color. Alternately use the quick colors on the top of the table, which show the six most recently used colors. Push a color to change the color of selected table cells.

It is possible to show two tables side by side. To show a second table use the >>> button. Hide the second table with the <<< button. When using side by side table display the software will prevent you from selecting the same table for both displays.

![Figure 14: Table display showing two tables at once](image)

1. **Operational condition**

The software will show the current operational condition as a red box drawn on the table. The history of the operation condition is also shown with a series of thin boxes drawn on the table. The age of the history information is set with **Trail**.

2. **Saving and Opening tables**

Use the **Save** and **Open** buttons at the top right to save or open the table as a human readable text file. Normally tables are saved to configuration files just like all other configuration information (see section V.L); however, saving to a table text file is useful because it stores the table suitably for use in a spreadsheet, allowing complex manipulations of the table data.

3. **Table indices**

The row and column indices are specific to the type of table. The index types are:

- **Throttle / Load**: *either* percent throttle position or the percent ratio of manifold to barometric pressure, based upon the table’s **MAP** selection.
• RPM: engine speed in revolutions per minute.
• CHT cold: Coldest cylinder head temperature in Celsius.
• CHT: Best estimate of cylinder head temperature from all CHT sensors in Celsius.
• CHT hot: Hottest cylinder head temperature in Celsius.
• MAT: Manifold air temperature in Celsius.
• Fuel pressure: gauge fuel pressure in kilopascals.
• Voltage: voltage of the injector power rail.
• Throttle: percent throttle position.
• Revolutions: number of engine revolutions since the engine started running.
• Density ratio: Ratio of the air density to standard day density.

4. Fuel table

The fuel table gives the amount of fuel to be injected, in units of micro-grams of fuel per revolution, per cubic centimeter of engine displacement. The output of the fuel table is the amount of fuel to inject for standard day conditions, with all fuel multipliers at 1.0. The actual fuel injected will be adjusted based on the pressures, temperatures, and fuel multipliers. Note that the fuel table gives the total fuel, how much goes to each injector depends on the injector ratio tables. The fuel table has rows indexed by throttle or load and columns indexed by RPM.

5. Fuel Multiplier table

This table provides a fuel multiplier computed as multiplier(k) = (1 + k), where k is the value from the table. It is intended to be used for engine-specific fueling adjustments. Adjusting this table (rather than the Fuel table) makes it more convenient to use a fleet calibration while keeping engine specific changes localized to this table. This table has rows indexed by throttle or load and columns indexed by RPM.

6. Spark Advance table

The spark advance table gives the amount of crank rotation, in degrees before top dead center, when the spark 1 output should be triggered. The spark advance table has rows indexed by throttle or load and columns indexed by RPM. Since spark 2 and 3 are referenced from spark 1, changing the spark advance table affects all spark outputs equally.

7. Spark 2 and 3 delay tables

The spark delay tables give the amount of crank rotation in degrees after the spark 1 output when the spark 2 or 3 output should be triggered. The spark delay tables have rows indexed by throttle or load and columns indexed by RPM. Note that spark 3 will not be output unless enabled in the engine configuration.

8. Spark retard table

The spark retard table gives a reduction of the ignition advance in degrees. The spark retard table has rows indexed by throttle or load and columns indexed by the hottest CHT. The table affects the advance of all the sparks equally. This table provides a way to manage engine knock induced by high load or high head temperatures.
9. Charge temperature table
The charge temperature table is used to compute a fuel multiplier that depends on the cylinder head and manifold air temperatures. The fuel multiplier is computed in one of two ways, depending on the value of *CHT reference*.

If *CHT reference* is zero (old style, version 1.5 and earlier):

\[
multiplier(k) = \frac{288.15}{k \times (CHT - MAT) + MAT_{ABS}}.
\]

If *CHT reference* is not zero:

\[
multiplier(k) = \frac{k \times (CHT_{REF} - 15) + 288.15}{k \times (CHT - MAT) + MAT_{ABS}}.
\]

This fuel correction accounts for air temperature and inlet tract heating affecting the amount of air trapped in the combustion chamber. The charge temperature table has rows indexed by throttle or load and columns indexed by RPM.

10. Manifold temperature table
The manifold temperature table gives a fuel multiplier computed as \( multiplier(k) = (1 + k) \), where \( k \) is the value from the table. The table has rows indexed by throttle or load and columns indexed by MAT. The output of this table replaces the charge temperature fuel multiplier. Choosing the charge temperature model versus the manifold temperature model is done by disabling or enabling this table respectively.

11. Cylinder head temperature table
The head temperature table gives a fuel multiplier computed as \( multiplier(k) = (1 + k) \), where \( k \) is the value from the table. The table has rows indexed by throttle or load and columns indexed by CHT. This table can be used to add fuel specifically to manage unusual head temperatures, for example extra fuel for when the engine is still warming up, or if it is overheating.

12. Injector 2 and 3 ratio tables
The injector ratio tables give the percent of fuel that should be output by that injector. If the table value is 0, all the fuel is output by the other injectors. If the table value is 100%, all the fuel is output by that injector. The injector ratio tables have rows indexed by throttle or load and columns indexed by RPM. Injector 1 does not have a ratio table, its output is: 100% - (injector2ratio + injector3ratio).

13. Injector 1, 2, and 3 phase tables
The injector phase tables give the crank angle in degrees for the injector output. The angle can specify the start, middle, or end of the injection depending on the injector configuration data. The tables have rows indexed by throttle or load and columns indexed by RPM.

14. Injector 1, 2, and 3 trim tables
The injector trim tables give the amount of time in microseconds to add to the pulse sent to the injector to account for the difference in injector opening versus closing time. The trim should be
positive if the injector opens slower than it closes and vice versa. The tables have rows indexed by fuel pressure and columns indexed by voltage.

15. Starting fuel table

The starting fuel table gives a fuel multiplier computed as \( \text{multiplier}(k) = (1 + k) \), where \( k \) is the value from the table. The table has rows indexed by CHT and columns indexed by the number of engine revolutions since start. This table is used to provide extra fuel for starting purposes. Unlike all the other tables when the number of revolutions exceeds the last column index this fuel multiplier will be 1.0, no matter the value in the last column.

16. Transient down fueling table

The transient down fueling table gives a fuel multiplier that depends on the rate of change of the throttle \( (\Delta \text{throttle/}\Delta t) \), if the rate is negative (i.e. throttle decreasing). The fuel multiplier is computed in the following way: \( \text{multiplier}(k) = (1 + k \times \Delta \text{throttle/}\Delta t) \), where \( k \) is the value from the table and \( \Delta \text{throttle/}\Delta t \) is the rate of change of the throttle in percent per second. The transient down fueling table has rows indexed by throttle or load and columns indexed by RPM.

17. Transient up fueling table

The transient up fueling table gives a fuel multiplier that depends on the rate of change of the throttle \( (\Delta \text{throttle/}\Delta t) \), if the rate is positive (i.e. throttle increasing). The fuel multiplier is computed in the following way: \( \text{multiplier}(k) = (1 + k \times \Delta \text{throttle/}\Delta t) \), where \( k \) is the value from the table and \( \Delta \text{throttle/}\Delta t \) is the rate of change of the throttle in percent per second. The transient up fueling table has rows indexed by throttle or load and columns indexed by RPM.

18. 2nd order density fuel correction table

The 2nd order density correction table is used to compute a fuel multiplier that depends on the density. This table is not used for the primary pressure and temperature correction; it is instead used as a 2nd order correction to remove any residual fueling error after the primary pressure and temperature fuel multipliers. The fuel multiplier is computed in the following way: \( \text{multiplier}(k) = (1 + k \times (1.225 - \text{density})/1.225) \), where \( k \) is the value from the table and density is the air density in kilograms per cubic meter. The 2nd order density correction table has rows indexed by throttle or load and columns indexed by RPM.

19. Density spark table

The density spark table is used to compute a spark advance adder that depends on the air density. Spark advance typically needs to increase as charge becomes less dense due to decreasing density (increasing altitude). The advance adder is computed in the following way: \( \text{adder}(k) = k \times (1.225 - \text{density})/1.225 \), where \( k \) is the value from the table and density is the air density in kilograms per cubic meter. The density spark table has rows indexed by throttle or load and columns indexed by RPM.
20. Ignition dwell table

The ignition dwell table gives the amount of time the ignition output is in the inactive (non-spark) state between spark events. The dwell time is used to compute the time the spark should be active, by subtracting the dwell time from the crank period. If the computed active time is less than the minimum active time from the engine configuration, the dwell will be reduced to honor the minimum active time. If the table is disabled the ignition active time will be the minimum active time. The ignition dwell table has rows indexed by voltage and columns indexed by RPM.

21. Minimum throttle table

The minimum throttle table gives the minimum throttle opening in percent. This table is only useful if IntelliJect is driving the throttle. The minimum throttle table has rows indexed by the coldest CHT and columns indexed by RPM. This table is useful for adjusting the idle speed of the engine for cold versus hot conditions.

22. Minimum throttle2 table

The second minimum throttle table gives a second minimum throttle output in percent. This table is only useful if the EFI is driving the throttle. The actual minimum throttle is the larger of this table and the first minimum throttle table. The table has rows indexed by the coldest head temperature and columns indexed by the density ratio. This table provides a way to adjust the minimum throttle for altitude and temperature.

23. Maximum throttle table

The maximum throttle table gives the maximum throttle opening in percent. This table is only useful if IntelliJect is driving the throttle. The maximum throttle table has rows indexed by the hottest CHT and columns indexed by RPM. This table provides a way to limit the power output of the engine if it is running hot (or cold, or fast). It also provides a way to limit the load on the engine at low speeds.

24. Maximum throttle2 table

The second maximum throttle table gives a second maximum throttle output in percent. This table is only useful if the EFI is driving the throttle. The actual maximum throttle is the smaller of this table and the first maximum throttle table. The table has rows indexed by the hottest head temperature and columns indexed by the density ratio. This table provides a way to limit the power output of the engine as a function of altitude and temperature.

25. Manifold pressure estimate table

The manifold pressure table gives a way to estimate the manifold pressure when the manifold pressure sensor is failed or disabled; or conversely, a way to estimate throttle when the manifold pressure sensor is working. The table gives the ratio of the manifold pressure to the barometric pressure, and has rows indexed by throttle (not load) and columns indexed by RPM.
26. Shaft power estimate table

The shaft power table provides an estimate of the shaft power, in Watts, on a standard dry day. The shaft power reported in the telemetry is computed from this table, using a mechanical efficiency estimate of 85% as well as the air density ratio, and the fuel multiplier from the charge temperature table. The shaft power table has rows indexed by throttle or load and columns indexed by RPM. This table does not affect the operation of the engine, it is used to provide telemetry data to the user.
C. Throttle

The throttle configuration defines how IntelliJect senses and/or commands the throttle position.

1. Throttle commands

If EFI drives throttle is selected IntelliJect can receive a throttle position command from pulse width modulation (PWM) input, from analog input, from the RPM controller, or from a throttle command message. Throttle commands from PWM or analog require that Enable PWM command or Enable analog command is selected. Both the analog and PWM signals can be enabled at the same time, and If both signals are valid the software will give precedence to the PWM. When IntelliJect boots up, if there is no valid throttle command, the throttle command is set to the Start Throttle configuration value.

The throttle command (PWM, analog, or user message) is passed through a rate limiter, given by the Rate Limit configuration value. The limiter gives the maximum rate at which the throttle command can move in percent throttle per second. In Figure 15 the rate limit is configured to be 400% per second, indicating that the throttle command will take ¼ second to transit from closed to open (or vice versa). Use a Rate Limit of zero to disable the limiter. The rate limited throttle command is then passed through a first order low pass filter with cutoff frequency given by the Command Filter configuration value; use zero to disable the filter.
If **Enable throttle curve** is selected the filtered throttle is passed through a throttle curve, which translates the input throttle to an output throttle. It is useful for affecting the “drivability” of the engine, by changing the sensitivity to throttle command changes. This is most commonly used to linearize the relationship between throttle command and engine power. The curve is defined by the table in the middle of the throttle configuration tab. Enter numbers in a cell or double-click a cell to change values of the table; and use the + or - to add or remove points in the curve (up to a maximum of 15 points). You can see the curve plotted on the right side of the window.

In Figure 15 the throttle command source is “Governor” indicating that the RPM controller is driving the throttle. The output of the governor 13.0%, and the throttle curve is converting that to a throttle output signal of 21.0%. After going through the throttle curve, the throttle output is limited to the min and max throttle as given by the throttle limit tables, see section V.B.21.

2. Throttle calibrations

The PWM input is calibrated with the *Closed PWM in* and *Open PWM in* configuration values. These respectively give the 0% and 100% throttle input pulse width in microseconds. If the *PWM in* calibration values are 0 the *PWM out* values are used instead. Calibration of the analog signal comes from the analog TPS sensor configuration, see section V.I.

The throttle output signal is converted to a 50Hz pulse width modulation (PWM) signal to drive a servo that actuates the throttle body. The output PWM is calibrated with the *Closed PWM out* and *Open PWM out* configuration value. If *CE CAN Address* is non-zero the PWM signal is also sent over the CAN bus to the Currawong Engineering CAN servo that has the node identifier *CE CAN Address* (1 to 254).

3. Start throttle limiting

The *Max Start* setting is used to protect against starting the engine with the throttle too high. If *Max Start* is greater than zero IntelliJect will not fire the spark or injector if the throttle was higher than *Max Start* before the engine began running. In that case an error is asserted, and you must lower the throttle below the *Max Start* value before the engine can be started.

Limiting the starting throttle can protect the engine (and the operator) from unintentional high power starts. In addition, you can intentionally raise the throttle above the *Max Start* value and crank the engine in order to clear a flood.

4. Throttle position sensing

The throttle used for table lookups comes from the throttle position sense (TPS).

- **Enable PWM TPS** should be selected to use the PWM input signal for TPS.
- **Enable analog TPS** should be selected to use the analog input signal for TPS.
- **Enable CE CAN TPS** should be selected to use the CAN throttle position report for TPS.
- **Enable MAP TPS** should be selected to use reverse lookup of the MAP estimate table with manifold and barometric pressure to compute the TPS. The MAP estimate table must be enabled and populated.
If multiple TPS signals are enabled the signals are given the following priority: PWM, Analog, CAN, MAP. If EFI drives throttle is selected the TPS is taken from the throttle output if no valid TPS is available.

Calibration of the PWM TPS and analog TPS are the same as the PWM command and analog command. Note that you cannot have both TPS and command enabled on the same input.

If IntelliJect is driving the throttle and a TPS is available IntelliJect will compute the error between the commanded throttle and the measured TPS. This difference is filtered by Command Filter and if it is greater than or equal to TPS Threshold an error is set (unless TPS Threshold is zero). The TPS error is included in the slow telemetry and is displayed on this window.

IntelliJect can run a feedback loop that adjusts the throttle output so the TPS matches the throttle command. This is done using TPS Int Gain which sets the integral feedback gain from the TPS error to the throttle command output adjustment (use zero to disable the feedback loop). The output of this loop is limited to TPS Threshold (or 10% if TPS Threshold is zero). Using TPS Int Gain is not a replacement for proper PWM output calibration, it is useful only for small throttle errors.

5. Building throttle curves

The typical use case for a throttle curve is to translate the input throttle to an output throttle that yields a power output which is proportional to the input throttle. In most cases IntelliJect is used to control engines driving fixed pitched propellers. Under static conditions (no forward speed) a fixed pitch propeller will absorb power in proportion to the cube of its RPM. In this case running the engine at different throttles and recording the resulting RPM provides enough information to compute a curve that performs the desired translation. You can use the Build... button to do this. The data that go into the computation are defined by strip-charts, which specify the time range from which the throttle and RPM information should be extracted, see Figure 16.

![Figure 16: Strip chart showing results of an engine checkout, defining the time range for the throttle curve build.](image)
Once you have defined a time window push the **Build...** button. If there is insufficient throttle motion in the data a warning is given.

![Insufficient data](image)

*Figure 17: Warning dialog indicating insufficient data to build a throttle curve.*

The curve builder looks through the dataset of throttle and RPM to find samples where the throttle was constant for at least one second. All such samples are averaged to produce a smaller data set which has discrete throttle and RPM pairs from which the curve can be built. For example, the data in Figure 16 contains 11 unique throttle output positions, resulting in 11 throttle and RPM pairs to build the curve from.

![Throttle Curve builder](image)

*Figure 18: The curve builder window showing input data and a computed throttle curve.*

The curve builder creates a regression fit from the throttle output to the RPM. The regression is a weighted cubic fit with a smoothness factor (from 0 to 100) controlled by the **Smoother** value. The regression is used to compute the required output throttle for any desired RPM. Simultaneously the builder either creates or loads an expected RPM curve; which defines the relationship between the input throttle command and the expected RPM. Finally, the curve builder computes the throttle curve such that each input throttle command generates an output throttle that yields the expected RPM.
If **Use expected RPM Table** is clear (as in Figure 18) the expected RPM is automatically computed according to a cubic relationship between RPM and throttle input command, with the maximum RPM of the input data anchoring the top of the curve. The expected RPM curve is in blue.

If **Use expected RPM Table** is checked (as in Figure 19) the expected RPM is defined by the table, which can be edited to specify a desired relationship between throttle input command and RPM. The expected RPM can be specified with just one point, or many points. The curve builder will fill out the expected RPM curve attempting to follow a cubic relationship between RPM and throttle input command. The expected RPM table can be loaded from, or saved to, a file; using the **Open...** and **Save...** buttons. Use the – and + buttons to remove or add rows to the expected RPM table.

In Figure 18 and Figure 19 the red line represents the curve fit, the red dots are the input points from the throttle and RPM dataset, the blue line is the expected RPM, and the green points are the values that go into the throttle curve. Use **# Curve Points** to change the number of points in the throttle curve, from 2 to 15.

If the curve builder was able to solve for the regression you can select **OK**, which will transfer the curve to the throttle configuration window, updating IntelliJect. Selecting **Cancel** will discard the curve.
D. Governor

The RPM governor drives the throttle output to achieve a commanded RPM\(^4\). The governor can only be used if Intelliject is driving the throttle, see section V.C. The RPM command comes from one of two sources: a direct RPM command received over a communications interface; or, an RPM command that is inferred from a throttle command. In Figure 20 the RPM was directly commanded to 2000, so the top of the page says: “Governor: 2000 From User”.

\[\text {Figure 20: RPM governor settings, and chart of RPM control performance under direct command.}\]

1. RPM and throttle model

The selection \textbf{Throttle commands RPM} selects if the RPM controller is driven by the throttle command, or only by a direct RPM command. When \textbf{Throttle commands RPM} is selected any throttle command will engage the RPM controller (direct RPM command is still allowed). Figure 21 gives an example of the RPM controller driven by throttle command.

\(^4\) The most common use case for the RPM controller is when the engine is driving a fixed pitch propeller, although it is not limited to that case.
Max RPM and Min RPM set the minimum and maximum RPM that can be commanded by either method. In addition, these limits are also part of the transfer function from throttle command to RPM command. The Throttle at Min RPM and the RPM to throt power complete the function, which can be seen plotted on the governor settings page. Throttle commands RPM also uses this function to determine the RPM command based on the throttle command. It is also used to determine the throttle output for direct RPM commands when the engine is not yet started.

![Figure 21: RPM governor driven by throttle command, 10% to 80% and back.](image)

2. Throttle output

The Update rate sets the rough frequency at which the RPM controller runs. The controller has an opportunity to run on each crank revolution, but if the elapsed time from the previous run is faster than the Update rate the controller will wait. It counts the time of successive revolutions until the elapsed time is longer than the Update rate and then computes the RPM using the entire elapsed time to get a less noisy reading for the control law.

The output of the RPM controller is a throttle that is subject to and min and max throttle tables (section V.B.21). The governor output is also subject to a low pass filter, with a cutoff frequency given by Filter (zero disables the filter). To further limit the throttle noise the filter applies a Softband. When the difference between the governor output and the previous filtered output is less than Softband the low pass filter cutoff frequency is reduced even further.

Filtering the output throttle is used to reduce the effect of noise, so that the throttle motion commanded by the governor is smoother. However, filtering adds lag to the output and will limit the responsiveness of the RPM controller, potentially driving it unstable if the controller gains are too aggressive.
3. Command trajectory

The command trajectory defines how the RPM controller responds to commands. The RPM cmd filter sets the cutoff frequency of a low pass filter applied to the RPM command (zero disables the filter). There are two types of RPM controllers: a trajectory controller, which is used if Trajectory gain is non-zero, and a classical controller. The trajectory controller computes the desired RPM rate by multiplying the RPM error against the trajectory gain. The desired RPM rate is then limited according to the Max RPM Rate. In the case of the classical controller (trajectory gain set to zero) the Max RPM Rate is used to limit the rate of the change of the RPM command. Max RPM Rate can be zero, in which case no rate limit is applied.

4. Inner loop gain scaling

Engine response to throttle motion is non-linear, so feedback gains that work well at one RPM may not work well at other RPMs. Gain scaling is used to change the strength of the feedback based on the RPM. The inner loop gains are multiplied by \((2 \times \frac{\text{RPM}}{\text{RPM}_{\text{MAX}}})^K\); where \(K\) is the Gain power in Figure 20.

The inner loop gains are the feedback gains when the RPM is half of the maximum RPM. If Gain power is 0.0 the gains do not change with RPM. When the power is positive gains increase as the RPM goes up, and vice versa. For example, when the power is 1.0 the gains are doubled at maximum RPM; and when the power is -1.0 the gains are halved at maximum RPM. When tuning the RPM controller, choose gains that work well at half the maximum RPM. Then use the Gain power to adjust the control feedback up or down at high or low engine speeds if needed.

5. Feedforward gain

Both the trajectory and classical controllers use a feedforward term, which adds throttle motion based on the commanded RPM, rather than the error in RPM. The feedforward term is very powerful; and is typically only used when trying to achieve maximum controller performance. The feedforward term is computed by multiplying \((0.01 \times \text{For gain})\) against the throttle predicted by the RPM and throttle model.

6. Derivative time constant

Both the classical and inner loop controllers must estimate the rate of change of RPM. A simple derivative of successive RPM measurements is too noisy, so the RPM controller implements a time history of RPM measurements and computes the rate of RPM change by using all the measurements made in the last Der interval time. Increasing the derivative interval reduces noise in the measurement, but adds lag to the controller, limiting performance.

7. Inner loop gains of the trajectory controller

The trajectory controller works by driving the throttle to achieve the desired rate of change of RPM as given by the trajectory gain and the RPM error. Pro gain and Int gain are the gains for proportional and integral feedback of the error in the rate of RPM; the Der gain is not used by the trajectory controller. The trajectory controller can do a better job of managing RPM overshoot or undershoot, which is why it is the default controller. However, trajectory controllers
can be harder to tune due to the close interaction between the trajectory gain, the maximum rate, and the inner loop gains.

8. Inner loop gains of the classical controller

The classical controller implements a conventional proportional, integral, derivative feedback controller using the Pro gain, Int gain, and Der gains. The derivative measurement is performed according to the derivative time interval.

9. Limiting throttle according to max and min RPM

When commanding the throttle directly (i.e. the governor is not running) you can use the governor to override the throttle command in order to prevent RPM from exceeding the Min RPM or the Max RPM. To do this you must be using the trajectory controller, and set the option Enable Low RPM Limiter and/or Enable High RPM Limiter. The governor monitors the rate of change of the RPM and will override the throttle output if the rate of change exceeds the allowable rate of change computed from the RPM limit and the trajectory gain. This feature is very useful for high altitude operation where the minimum throttle setting may be too low for reliable engine operation.

Figure 22: RPM limiter, without and with throttle rate limiting.

Figure 22 demonstrates the limiter in action. The throttle was commanded from 0% to 100% and back twice. The throttle output follows the command until the limiter overrides it to prevent the RPM from exceeding the limits. In the first example the throttle rate limit was set to zero, in the second example the rate limit was set to 200%/s.
10. Default governor settings

You can set the RPM governor to default values by setting the max rpm less than the minimum, or by setting the max rpm to be 0. IntelliJect will recognize this as invalid and overwrite all the governor settings with the default values.

11. Using the throttle curve with the RPM controller

The output of the RPM governor goes through the throttle curve table, just as a normal throttle command does. The throttle output curve (see section V.C.1) is typically used to improve the “drivability” of the engine, by linearizing the sensitivity to throttle commands. In just the same way that the throttle curve can make the engine more tractable for a user, it can do the same thing for the RPM controller.
E. Engine

This page gives configuration options about the engine including the how the crank sense is configured, and how the ignition is operated.

![Engine Configuration Diagram]

Figure 23: Engine configuration

1. Engine

- **Ignore input enable** instructs IntelliJect to ignore the status of the input enable line. This is useful only for testing. Note that the CDI ignition outputs will not activate if the input enable is not high.
- **Displacement** is the engine displacement in cubic centimeters. This value scales the fuel injected.
- **CHT reference** is the cylinder head temperature used as the reference point for the charge temperature correction. This value should be set equal to the head temperature that was used during the determination of the fuel table. If CHT reference is zero the old style charge temperature is correction is used (from software version 1.5 and earlier).
- **MAP threshold** sets maximum difference between the manifold pressure measurement and the manifold pressure estimate before the MAP error asserts and the measured pressure is replaced with the estimate. Set the threshold to 0 (or disable the MAP estimate table) to disable MAP error detection.
- **RPM filter** is the cutoff frequency of the first order low pass filter applied to the RPM measurement. Each crank rotation yields a new engine speed measurement, which is fed into the filter to compute the RPM measurement. 0 disables the filter.
• **Hard rev limit** is the maximum engine speed before IntelliJect stops all spark and injector outputs. Engine speeds above this limit trigger the hard overspeed error. Set the rev limit to 0 to disable rev limiting.

• **Soft rev limit** is the maximum engine speed before IntelliJect begins interrupting spark and injector outputs. Engine speeds above this limit trigger the soft overspeed error. Engine speeds between the soft and hard rev limit result in a periodic interruption of the spark and injection in proportion to the amount of overspeed. For example, if the engine speed is halfway between the soft and hard limit every other spark and injection event will be skipped.

2. **Dual crank sense options**

• **Prevent normal rotation** stops spark and injector outputs when the engine is rotating in normal direction. This option will have no effect if direction cannot be determined from the crank sensors. Engine rotation is defined by the forward and reverse crank sense angles, or by the normal and reverse gap spacing on the crank wheel.

• **Prevent reverse rotation** stops spark and injector outputs when the engine is rotating in reverse direction. This option will have no effect if direction cannot be determined from the crank sensors.

• The direction of rotation is displayed as the bottom of this box; as one of “Normal Direction”, “Reverse Direction”, or “Direction Unknown”.

• If both crank sensors are enabled and healthy the angle of the crank (according to sense 1) at which sense 2 synchronizes is displayed as “Crank 2 angle: angle/deviation”. This value can be compared to the expected crank 2 angle to estimate the error in the crank sensor alignment.

3. **Crank Sense 1 and 2**

There are two crank sense inputs (1 and 2), which operate independently to provide redundant crank speed and position sensing. Crank sensing can be done using a once-per-revolution sensor, or a crank wheel that provides multiple sense events per revolution. The style of sensor, and the design of the crank wheel, can be different between sense 1 and sense 2; IntelliJect will automatically determine the optimum trigger source to fire each output. This determination is made based on the health of the crank sensors, and the delay time between the closest crank sense event and the start of the output.

Once-per-revolution sensors should be placed as close as reasonable to the earliest spark output of the cycle; typically, crank angle 320° (40° before top dead center). If you use two once-per-revolution sensors, and if they are placed at different crank angles, IntelliJect will compute the direction of rotation if both sensors are healthy. To do this the *Normal* and *Reverse* angles must be correctly specified for both sensors (and they must not be 180° apart).

Crank wheels use evenly spaced teeth, with one or more teeth missing to synchronize the wheel position. Crank wheels can be directional, which means they have two gaps of different sizes; and IntelliJect will use the gap timing to determine the direction of rotation. When using a crank wheel, the *Normal* and *Reverse* angles give the crank shaft angle of the first tooth after the larger of the gaps, i.e. the Big Gap. When a crank wheel has two gaps the *Small Gap* should be placed
asymmetrically from the Big Gap, so that the pattern of gap timing is dependent on the direction of rotation. Figure 24 gives an example crank wheel with a total count of 20, a big gap of 2, a small gap of 1, normal gap spacing of 3, and reverse gap spacing of 14. Because the big gap is symmetrical with respect to TDC the wheel in Figure 24 has a reverse and normal angle of 207°.

Figure 24: Annotated diagram of a directional crank wheel with a count of 20

If you are using a crank wheel with only one gap IntelliJect will determine rotation direction by comparing the synchronization time of sense 1 and sense 2, just like once-per-revolution sensors. It is also possible to mix a crank wheel with a once-per-revolution sensor.

- **Enable Sensor** should be checked to enable the crank sensor. You should disable any crank sensor which is not being used; otherwise IntelliJect will report errors for that sensor.
- **Prefer Sensor** should be checked to tell IntelliJect that sensor should be used to schedule the spark and injector outputs. You can only have one preferred sensor. If no sensor is preferred IntelliJect will select the sensor which is which closest to the start of the output. IntelliJect will ignore the preferences if a crank sensor had an error; instead using the sensor which does not have an error (or whose last error is further in the past).
- **Normal angle** is the crank shaft angle, in degrees of normal crank rotation after top dead center (TDC), when the sensor becomes active. For crank wheels this is the first tooth after the Big Gap.
• **Reverse angle** is the crank shaft angle, in degrees of reverse crank rotation after top dead center, when the sensor becomes active. For crank wheels this is the first tooth after the **Big Gap**.

• **Active High** indicates that the crank sense event occurs on the rising edge of the crank sense signal, otherwise the falling edge is used for the crank sense.

• **Sense delay** is the microseconds of time delay caused by filtering on the crank sense signal. It is typically zero.

• **Wheel** enables the crank wheel decoding logic. You will not be able to enable the crank wheel if the **Total count** and **Gap** settings are inconsistent. Clear **Wheel** for a once-per-revolution sensor.

• **Total count** gives the number of virtual teeth on the crank wheel. This is the sum of the actual teeth and the missing teeth. This value must be more than **Big Gap + Small Gap**.

• **Big Gap** gives the number of teeth missing in the big gap. This value cannot be zero if the crank wheel is enabled.

• **Small Gap** gives the number of teeth missing in the small gap. This value must be less than **Big Gap**. If it is zero, there is only one gap and the wheel is not directional.

• **Normal Space** gives the number of teeth between the **Big** and **Small Gap** in the normal direction of rotation. If **Normal Space** is zero the wheel does not have a small gap, no matter the value of **Small Gap**.

• **Reverse Space** gives the number of teeth between the **Big** and **Small Gap** in the reverse direction of rotation. **Reverse Space** and **Normal Space** are redundant with each other. If either value is given as zero IntelliJect will compute it as **Total count - Big Gap - Small Gap - Space**.

The state of the crank sense inputs is part of the slow telemetry and turns the indicator light (next to the **Active High** checkbox) green when active. When the engine is running these will be changing too fast to be useful, but if the engine is not running these indicators are helpful for measuring the crank sense angles.

At the bottom the crank sense window are six indicator lights (S1, S2, S3, I1, I2, I3) which specify which spark and injector outputs are active and which crank sensors are triggering them. In Figure 23 spark 1 and 2 and injector 1 are being triggered by crank sense 2.

4. **Ignition**

IntelliJect supports three ignition outputs.

• **Ignition Active High** should be selected to trigger the ignition with a rising edge, and vice versa. Most CDI ignitions are triggered with a rising edge

• **Min Active Time** sets the minimum microseconds of time the ignition output should be in the active state when the spark is fired. IntelliJect will compute the active time based on the ignition dwell time. However, If the ignition dwell time table is disabled, or if the dwell is too long to fit in the crank period, the ignition output will be active for **Min Active Time**.

• **Max Dwell Time** sets the maximum amount of time, in milliseconds, the ignition output can be in the inactive state. This is important if the ignition output is used to control a coil ignition. A coil ignition is energized when the ignition output is inactive; and leaving it energized can
damage the coil or lead to excessive power consumption. The largest *Max Dwell Time* is 255 ms (minimum 235 RPM). IntelliJect will not be able to maintain normal ignition operation if the engine speed falls below the value that corresponds to the *Max Dwell Time*. Setting *Max Dwell Time* to zero will disable this feature.

- **Enable Spark 3** should be selected to enable the third spark output. The third spark output uses the same IO line as the heartbeat signal; if it is enabled the heartbeat signal is disabled.
F. Cooling

IntelliJect outputs two signals to control cooling. It is expected these signals will be connected to a cowl flap servo or electronic cooling fan speed controller. If the engine is not running, the cooling output percentage matches the throttle output (this makes it easy to verify the cooling system functionality before engine start). If the engine is running the cooling output is determined from a feedback control law based on the head temperatures.

1. Cooling Feedback

- **Enable split cooling** should be set to run cooling output 1 independently of cooling output 2. Independent cooling will only take place if both CHT sensors are healthy. If cooling output is not split the hottest CHT sensor is used to drive the cooling outputs.
- **CHT too cold** is the cylinder head temperature of the coldest sensor below which a head temperature error will be asserted.
- **CHT desired** sets the command for the control loops. Cooling output will increase or decrease as needed to make CHT equal to **CHT desired** if the engine is running.
- **CHT too hot** is the cylinder head temperature of the hottest sensor above which a head temperature error will be asserted.
- **Trajectory gain** sets the desired rate of change of the CHT based on the error between **CHT desired** and the measured CHT. If Trajectory gain is 0 the control law reverts to classic PID.
- **Forward** is the feed forward gain from percent throttle to percent cooling. Note that if the head temperature sensors are disabled or failed, the cooling output will be entirely determined by this gain when the engine is running.
- **Proportional** is the proportional feedback gain. If **Trajectory gain** is zero (classic PID) this gain goes from the head temperature error to cooling output in percent. If **Trajectory gain** is non-zero this gain goes from the head temperature rate error to cooling output in percent.
- **Integral** is the integral feedback gain. If **Trajectory gain** is zero (classic PID) this gain goes from integral of the head temperature error to cooling output in percent. If **Trajectory gain** is non-zero this gain goes from integral of the head temperature rate error to cooling output in percent.
- **Der gain** gives the derivative feedback gain (only if **Trajectory gain** is zero) that goes from head temperature rate error to cooling output in percent.
- **Der interval** gives the time span used to compute the rate of change of head temperature. Longer time spans have less noise but more lag.
2. Cooling output 1 and 2

IntelliJect outputs two 50Hz PWM signals to control cooling. Each output has the same configuration options. Note there is no physical PWM output for cooling 2 for IntelliJect hardware prior to rev 3.

- **PWM min** is the pulse width in microseconds for 0% cooling.
- **PWM max** is the pulse width in microseconds for 100% cooling. This value may be less than **PWM min**, depending on the requirements of the cooling system. If this value is equal to **PWM min** the output will be fixed at **PWM max**.
- **CE CAN Servo** is the node ID (1 to 254) to send the cooling PWM command to a Currawong Engineering CAN servo. Use 0 to disable the CAN servo output.
- **Output** displays the current cooling output from the IntelliJect telemetry.
G. Injectors

Injectors gives injector configuration data, as well as telemetry details about the operation of the injectors. IntelliJect supports up to three injectors (rev3 hardware and later), each of which has its own configuration data. Injector 1 cannot be disabled. Injectors that are not used should be disabled to suppress nuisance errors.

![Injectors configuration and telemetry](image)

- **Output multiplier** is a convenient way to change the total fuel flow without affecting other settings. It is typically left at 1.0.
- **Used multiplier** changes the reported fuel flow and fuel used. It is typically left at 1.0 unless it needs to be adjusted to correct errors in the reported fuel flow. This will not affect injector operation, it will only affect the telemetry.
- **Pressure** sets the nominal fuel pressure in kilopascals at which the injector flow rates apply. IntelliJect will adjust the injector opening time to account for fuel pressure deviation from this value.
- **Volume per MAT** gives the percentage change in fuel volume per degree Celsius manifold air temperature. This value gives the fuel volume change for temperatures that are not standard day (15°C), under the assumption that the injector is a volume flow device, but the goal is to control the mass of the fuel injected.
- **Phase Edge** specifies what edge of the injector output is controlled by the injector phase tables: Start, Middle, or End.
- **Independent operation** should be selected to run the injectors with independent charge temp and head temp fuel multipliers. If selected the multipliers become injector specific using independent CHT and MAT sensors, see below for more details.
- **Spare temp is CHT3** should be selected to consider the spare temperature sensor as a third CHT sensor.
- **Spare temp is MAT2** should be selected to consider the spare temperature sensor as a second manifold temperature sensor. If both **Spare temp is CHT3** and **Spare temp is MAT2** are selected IntelliJect will clear **Spare temp is MAT2**.

- **Flow rate** gives the full open fuel flow rate of the injector(s) in grams per minute of fuel, at the nominal fuel pressure, at 15C MAT. This number is used to determine how long to open the injector. Setting the flow rate to zero disables the injector.

- **Min pulse** is the minimum amount of time the injector(s) can be opened in microseconds. Injector opening times less than this minimum may result in unpredictable injector performance. If the computed injector time is less than the minimum IntelliJect will skip the injection event, and instead add that events injector time to the next event.

- **Injector 2** must be selected to enable the second injector output. **Flow rate** and **Min pulse** of the second injector work the same as they do for the first injector.

- **Injector 3** must be selected to enable the third injector output. **Flow rate** and **Min pulse** of the third injector work the same as they do for the first injector.

- The **Reset Fuel Used** button commands IntelliJect to reset the accumulated fuel burn that is reported in telemetry.

1. Choosing temperatures

   There are four possible temperature sensors, a dedicated MAT, two dedicated CHTs and a spare temperature that can be configured as a second MAT or a third CHT based on the injector configuration options **Spare temp is CHT3** and **Spare temp is MAT2**. From these sensors IntelliJect will choose and compute a hottest, coldest, average, and injector specific (1, 2, and 3) head temperatures. The choice of temperatures depends on sensor health (i.e the sensor is enabled and its reading is within sensor limits). For example, if only a single CHT sensor is healthy all the head temperatures will be set to that one value. If two CHTs are healthy the remaining CHT will be the average of the two that are healthy.

   IntelliJect will also choose and compute an average manifold temperature and injector specific manifold temperatures. Unlike the head temperature there are only two possible manifold temperature sensors, but the same sensor health rules apply: if only one manifold temperature is healthy all the manifold temperatures will be set to that one value. If both MATs are healthy the third manifold temperature will be set to the average manifold temperature.

2. Computing fuel to be injected

   To compute the fuel injected on each revolution the system performs the following steps:

   a) Compute the engine load according to the configuration: either the throttle percentage or $100 \times \frac{\text{MAP}}{\text{Baro}}$.

   b) Compute nominal fuel based on the pressure, fuel, and fuel multiplier tables.

   $$\text{Fuel}_{\text{NOM}} = \left(\frac{\text{Baro}}{101.325}\right) \times 10^{-6} \times \text{fuel\_table}(\text{load, RPM}) \times \text{displacement}.$$  

   $$\text{Fuel}_{\text{NOM}} = \text{Fuel}_{\text{NOM}} \times (1 + \text{fuelmult\_table}(\text{load, RPM})).$$

   $10^{-6}$ converts from micro-grams to grams and 101.325 is standard day air pressure in kPa.
c) Choose manifold and head temperatures based on sensor health and the **Independent operation**, **Spare temp is CHT3**, and **Spare temp is MAT2** selection. If **Independent operation** is not checked then the temperatures used are the average values:

\[
\text{MAT}_{1\text{ST}} = \text{MAT}_{2\text{ND}} = \text{MAT}_{3\text{RD}} = \text{MAT}_{\text{AVERAGE}}
\]

\[
\text{CHT}_{1\text{ST}} = \text{CHT}_{2\text{ND}} = \text{CHT}_{3\text{RD}} = \text{CHT}_{\text{AVERAGE}}
\]

d) Compute the charge temperature fuel multiplier, which may be injector specific if **Independent operation** is checked. Compute the multiplier in one of two ways: the old way (if \(\text{CHT}_{\text{REF}}\) is zero, or software is older than 1.6), or the new way. The old charge temperature fuel multiplier is:

\[
k = \text{chargetemp\_table}(\text{load}, \text{RPM})
\]

\[
\text{FM}_{1\text{STCHARGE}} = \frac{288.15}{(k \times (\text{CHT}_{1\text{ST}} - \text{MAT}_{1\text{ST}})) + \text{MAT}_{1\text{ST}} + 273.15}.
\]

\[
\text{FM}_{2\text{NDCHARGE}} = \frac{288.15}{(k \times (\text{CHT}_{2\text{ND}} - \text{MAT}_{2\text{ND}})) + \text{MAT}_{2\text{ND}} + 273.15}.
\]

\[
\text{FM}_{3\text{RDCHARGE}} = \frac{288.15}{(k \times (\text{CHT}_{3\text{RD}} - \text{MAT}_{3\text{RD}})) + \text{MAT}_{3\text{RD}} + 273.15}.
\]

The new charge temp fuel multiplier is:

\[
\text{FM}_{1\text{STCHARGE}} = \frac{(k \times (\text{CHT}_{\text{REF}} - 15) + 288.15)}{(k \times (\text{CHT}_{1\text{ST}} - \text{MAT}_{1\text{ST}})) + \text{MAT}_{1\text{ST}} + 273.15}.
\]

\[
\text{FM}_{2\text{NDCHARGE}} = \frac{(k \times (\text{CHT}_{\text{REF}} - 15) + 288.15)}{(k \times (\text{CHT}_{2\text{ND}} - \text{MAT}_{2\text{ND}})) + \text{MAT}_{2\text{ND}} + 273.15}.
\]

\[
\text{FM}_{3\text{RDCHARGE}} = \frac{(k \times (\text{CHT}_{\text{REF}} - 15) + 288.15)}{(k \times (\text{CHT}_{3\text{RD}} - \text{MAT}_{3\text{RD}})) + \text{MAT}_{3\text{RD}} + 273.15}.
\]

e) Compute the manifold temp fuel multiplier, which replaces the charge temperature fuel multiplier. This replacement is only performed if the manifold temperature table is enabled. The enable or disable of the manifold temperature table selects which temperature compensation model you prefer.

\[
\text{FM}_{1\text{STCHARGE}} = 1 + \text{manifoldtemp\_table}(\text{load}, \text{MAT}_{1\text{ST}}).
\]

\[
\text{FM}_{2\text{NDCHARGE}} = 1 + \text{manifoldtemp\_table}(\text{load}, \text{MAT}_{2\text{ND}}).
\]

\[
\text{FM}_{3\text{RDCHARGE}} = 1 + \text{manifoldtemp\_table}(\text{load}, \text{MAT}_{3\text{RD}}).
\]

f) Compute the head temp fuel multiplier, which may be injector specific:

\[
\text{FM}_{1\text{STHEAD}} = 1 + \text{headtemp\_table}(\text{load}, \text{CHT}_{1\text{ST}}).
\]

\[
\text{FM}_{2\text{NDHEAD}} = 1 + \text{headtemp\_table}(\text{load}, \text{CHT}_{2\text{ND}}).
\]

\[
\text{FM}_{3\text{RDHEAD}} = 1 + \text{headtemp\_table}(\text{load}, \text{CHT}_{3\text{RD}}).
\]

g) Compute the transient fuel multiplier.

**Decreasing throttle:**

\[
\text{FM}_{\text{TRANSIENT}} = 1 + (\Delta\text{load}/\Delta t) \times \text{transientdown\_table}(\text{load}, \text{RPM}).
\]

**Increasing throttle:**

\[
\text{FM}_{\text{TRANSIENT}} = 1 + (\Delta\text{load}/\Delta t) \times \text{transientup\_table}(\text{load}, \text{RPM}).
\]
h) Compute the 2nd order altitude fuel multiplier, which depends on the density ratio.

\[
\text{Density}_\text{RATIO} = \frac{\text{Baro}}{101.325} \times \frac{288.15}{\text{MAT}_{\text{AVG}} - \text{ABS}}.
\]

101.325 is the standard day air pressure in kPa, and 288.15 is the standard day absolute temperature in Kelvin.

\[
\text{FM}_{\text{ALTITUDE}2} = 1 + (1 - \text{Density}_\text{RATIO}) \times 2^{\text{nd}} \text{Density}\_\text{table}(\text{load, RPM}).
\]

i) Compute the start fuel multiplier.

\[
\text{FM}_{\text{START}} = 1 + \text{startfuel\_table} (\text{CHT, \#revs}).
\]

j) Compute the fuel for each injector

\[
\begin{align*}
\text{InjFraction}_2 & = 0.01 \times \text{injector2ratio\_table}(\text{load, RPM}). \\
\text{InjFraction}_3 & = 0.01 \times \text{injector3ratio\_table}(\text{load, RPM}). \\
\text{InjFraction}_1 & = 1.0 - (\text{InjFraction}_2 + \text{InjFraction}_3). \\
\text{Fuel}_1 & = \text{Fuel}_{\text{NOM}} \times \text{InjFraction}_1 \times \text{FM}_{1\text{STCHARGE}} \times \text{FM}_{1\text{STHEAD}} \times \text{FM}_{\text{TRANSIENT}} \times \text{FM}_{\text{ALTITUDE2}} \times \text{FM}_{\text{START}}. \\
\text{Fuel}_2 & = \text{Fuel}_{\text{NOM}} \times \text{InjFraction}_2 \times \text{FM}_{2\text{NDCHARGE}} \times \text{FM}_{2\text{NDHEAD}} \times \text{FM}_{\text{TRANSIENT}} \times \text{FM}_{\text{ALTITUDE2}} \times \text{FM}_{\text{START}}. \\
\text{Fuel}_3 & = \text{Fuel}_{\text{NOM}} \times \text{InjFraction}_3 \times \text{FM}_{3\text{RDCHARGE}} \times \text{FM}_{3\text{RDHEAD}} \times \text{FM}_{\text{TRANSIENT}} \times \text{FM}_{\text{ALTITUDE2}} \times \text{FM}_{\text{START}}.
\end{align*}
\]

The fuel from the above is in units of grams per revolution, multiply by RPM to get fuel flow in grams per minute. Note start fuel multiplier is set to 1.0 if the number of revolutions is larger than the rightmost column of the start fuel table. The density and fuel multipliers are visible in the telemetry on the injectors window, see Figure 26.

3. Computing injector opening times

a) Injector pressure = fuel pressure + barometric – MAP.

b) Fuel pressure compensation:

\[
P_{\text{comp}} = \left(\frac{\text{nominal pressure}}{\text{injector pressure}}\right)^{0.5}
\]

c) Fuel volume compensation:

\[
V_{\text{comp}} = 1 + 0.01 \times \text{VolumePerMAT} \times (\text{MAT}_{\text{AVG}} - 15)
\]

d) Nominal first injector pulse in micro-seconds:

\[
\text{Pulse}_{\text{NOM}1} = 10^6 \times 60 \times P_{\text{comp}} \times V_{\text{comp}} \times (\text{Fuel}_1 / \text{flowrate}_1).
\]

e) Nominal second injector pulse in micro-seconds:

\[
\text{Pulse}_{\text{NOM}2} = 10^6 \times 60 \times P_{\text{comp}} \times V_{\text{comp}} \times (\text{Fuel}_2 / \text{flowrate}_2).
\]

f) Nominal third injector pulse in micro-seconds:

\[
\text{Pulse}_{\text{NOM}3} = 10^6 \times 60 \times P_{\text{comp}} \times V_{\text{comp}} \times (\text{Fuel}_3 / \text{flowrate}_3).
\]

g) First injector pulse:

\[
\text{Pulse}_1 = \text{Pulse}_{\text{NOM}1} + \text{injector1trim\_table}(\text{injector pressure, voltage}).
\]
h) Second injector pulse:
\[ \text{Pulse}_2 = \text{Pulse}_{\text{NOM}2} + \text{injector2trim_table}(\text{injector pressure}, \text{voltage}) \]

i) Third injector pulse:
\[ \text{Pulse}_3 = \text{Pulse}_{\text{NOM}3} + \text{injector3trim_table}(\text{injector pressure}, \text{voltage}) \]

The injector pressure of step a) represents the pressure difference across the injector spray plate. It is not exactly fuel pressure due to the assumption the fuel is injected into the intake manifold. Therefore, low manifold pressure increases the pressure across the injector spray plate. The square root of injector pressure in step b) adjusts the output to account for variations in fuel pressure. The fuel volume compensation in step c) assumes the injector temperature is the same as the manifold, and the fuel will take on this temperature as it flows through the injector.

4. Effect of injector minimum pulse

Injector minimum pulse logic is applied to the Pulse\textsubscript{NOM} value, rather than the Pulse value. This prevents adding multiple injector trims when skipping injections due to the minimum pulse logic. The injector time telemetry on the injectors window is the Pulse\textsubscript{NOM} value, see Figure 26. Anytime this display is less than the minimum pulse Intelliject is skipping injections.

The minimum pulse logic is useful for low load conditions where the amount of fuel needed is small. Injector skipping increases the dynamic range of the injector, increasing the number of injector choices for a given engine. It may seem counter intuitive that injection events can be skipped, but for small engines at low load there is a substantial delay between fuel injection and fuel reaching the combustion chamber, making injector skipping viable.
H. Pump

The fuel pump is controlled by a FET switch connected to the 12V rail. In Rev 3 hardware a 5V output is also provided for controlling external pump switches. The pump controller computes the duty cycle applied to the switch from 0% (completely off) to 100% (completely on). The duty cycle is computed by either a bang-bang controller, or by a continuous forward-proportional-integral controller, depending on the configuration. The current pump duty cycle is part of the telemetry and is displayed on the Pump window.

![Figure 27: Pump configuration.](image)

1. Pressures
   - **Command** is the desired fuel pressure in kilopascals the pump controller tries to achieve.
   - **Error margin** is the amount of fuel pressure error in kilopascals that it takes to trigger a fuel pressure error.
   - **On-off margin** defines the pressure in kilopascals for the bang-bang controller hysteresis. The bang-bang controller sets the pump to the minimum duty cycle when the pressure exceeds the command by the on-off margin; and sets the pump to maximum duty cycle when the pressure is below the command by the on-off margin. If the on-off margin is zero, the continuous controller is used instead of the bang-bang controller.

2. Gains
   - **Forward** sets the continuous fuel pump controller feed forward gain in units of pump duty cycle percentage per gram/minute of fuel flow.
   - **Proportional** sets the continuous fuel pump controller proportional feedback gain in units of pump duty cycle per kilopascal of pressure error.
   - **Integral** sets the continuous fuel pump controller integral feedback gain in units of pump duty cycle per kilopascal of pressure error per second.
   - **Rate limit** sets the maximum rate of change of the pump duty cycle in percent per second. The rate limiter is useful for managing the in-rush current of the pump. Use 0 to disable the rate limiter.
3. Logic

- **Max output if failure** causes the pump output to go to maximum if the fuel pressure sensor fails or is not enabled. If Max output if failure is clear a failed fuel pressure sensor will result in the pump output computed according to the feed forward gain. Do not to set this option unless a mechanical pressure regulator is included in the system.

- **RPM Enables pump** turns on logic that only enables the pump when the engine is running, except for the **Prime time**. When clear the pump will run anytime IntelliJect is enabled and the pressure is low.

- **Prime time** sets the number of seconds the pump can run to prime the fuel system when the engine is **not** running. Prime time only applies if RPM Enables pump is set. After the pump has run for Prime time the pump will turn off until the engine starts, or the IntelliJect is disabled and re-enabled.

4. Duty cycle

- **Minimum** sets the minimum pump duty cycle percentage for either controller.

- **Maximum** sets the maximum pump duty cycle percentage for either controller.

- **PWM period** sets the period of the pump output signal in microseconds.

![Figure 28: Pump duty cycle and fuel pressure with the bang-bang and continuous controller at constant throttle.](image)
I. Sensors

The sensor configuration and telemetry are on the sensors page. Like the tables page, the sensors page uses a drop down to select a single sensor out of the list of sensors to configure.

- **Enabled** must be selected to enable a sensor. If a sensor is not enabled its value is fixed at the *Nominal* setting.
- **Default** should be selected to force the sensor configuration to its default. You will not be able to change any other settings if this is set.
- **Autocorrect** should be selected to allow this sensor to be automatically corrected by IntelliJect. The only sensors which can be automatically corrected are the MAP, and the Total and 12V current sensors.
- **KTY84** should be selected to automatically apply a KTY84 nonlinear temperature sensor calibration. This is commonly used for CHT1 and CHT2 sensors. **KTY84** is only an option for MAT, CHT, or the spare temperature sensors.
- **KTY83** should be selected to automatically apply a KTY83 nonlinear temperature sensor calibration. This is commonly used for MAT sensors. **KTY83** is only an option for MAT, CHT, or the spare temperature sensors.
- **1000Ω RTD** should be selected to automatically apply the correction for a platinum resistive temperature detector (RTD) sensor that has a 1000 Ω resistance at 0°C. This is only an option for MAT, CHT, or spare temperature sensors
- **Sealed gauge** should be selected if the fuel pressure sensor is of the sealed gauge type. This is only an option for the fuel pressure sensor.
- **Filter** sets the cutoff frequency of the low pass filter applied to the sensors output. Use 0 to disable the low pass filter.
- **Minimum** sets the lowest valid measurement from the sensor. If, after applying the sensor calibration, the result is less than **Minimum** the sensor is considered invalid, an error is set, and the output is set to *Nominal*. 
• **Maximum** sets the highest valid measurement from the sensor. If, after applying the sensor calibration, the result is more than **Maximum** the sensor is considered invalid, an error is set, and the output is set to **Nominal**.

• **Nominal** sets the value used for a sensor when the sensor is disabled or is reading outside the minimum or maximum bounds.

• **Gain**... sets the constant that converts from raw sensor output (which goes from 0.0 to 1.0) to the units of the sensor. The gain is interpreted differently if **KTY84**, **KTY83**, or **1000Ω RTD** is set, see below for details. If you push the **Gain**... button you will be prompted to enter the actual sensor value, and IntelliJect will estimate the sensor gain that results in this value.

• **Offset**... sets the constant that is subtracted from the raw sensor output (which goes from 0.0 to 1.0) account for sensor bias. The offset is interpreted differently if **KTY84**, **KTY83**, or **1000Ω RTD** is set, see below for details. If you push the **Offset**... button you will be prompted to enter the actual sensor value, and IntelliJect will estimate the offset that results in this value.

1. **Sensor list**

The drop down pictured in Figure 29 gives the list of sensors that are available.

• Main Volts: input voltage applied to board. This is only used in telemetry, it does not impact engine operation.

• MAT: manifold air temperature sensor in Celsius. If the MAT sensor is invalid, and the OAT sensor is good, the OAT value will be used in place of the MAT.

• CHT1: first cylinder head temperature sensor in Celsius. If the CHT1 sensor is invalid, and the CHT2 sensor is good, the CHT2 value will be used in place of the CHT1.

• CHT2: second cylinder head temperature sensor in Celsius. If the CHT2 sensor is invalid, and the CHT1 sensor is good, the CHT1 value will be used in place of the CHT2.

• MAP: manifold air pressure sensor in kilopascals. If the MAP sensor is invalid or disabled, the barometric pressure multiplied by the output of the MAP estimate table will be used instead.

• Fuel pressure: fuel pressure sensor in kilopascals. The fuel pressure sensor is external to the IntelliJect board; it connects via the fuel sensor analog input on the harness.

• Analog TPS: analog throttle position sensor or throttle position command (depending on throttle configuration) in percent.

• CPU Temp: temperature of the processor in Celsius. This is only used for telemetry.

• Spare temperature: is a spare temperature sensor input that is setup for the same type of sensor as the MAT or CHT sensors.

• Digital Baro pressure: digital barometric pressure sensor in kilopascals, which is used if the analog sensor is disabled or invalid.

• Outside temp: outside air temperature (OAT) sensor in degrees Celsius. This sensor is used as a backup for the MAT.

• Main Current: input current in Amps.

• 12 Volt voltage of the nominal 12V rail used to drive the injectors and fuel pump.

• 12 Volt Current: current of the nominal 12V rail. This is only used in the telemetry.

• Analog Baro pressure: analog barometric pressure sensor in kilopascals.
2. Sensor sampling

The analog sensors are sampled at 1kHz (every 1.0 milli-seconds). Each sample is accumulated into a buffer until the crank sensor indicates another rotation of the engine has completed (or until 100 milli-seconds has elapsed). The accumulations are averaged and converted to a normalized raw sensor reading that goes from 0.0 to 1.0.

3. Conversion of normal sensors

The sensor output is computed according to Output = Gain × (Raw – Offset); where Raw is the value determined from the sensor sampling.

4. Conversion of KTY8x sensors

The resistance of the sensor is computed as resistance = R = ((3.3 × Gain × Raw) ÷ tempcurrent) - Offset × 1000. Gain and Offset are typically 1.0 and 0.0 respectively; and tempcurrent is a factory set constant that specifies the current in Amps from the temperature sensor driver (typically 0.00116). Notice that Offset is treated as a resistance offset in kΩ.

Temperature is computed according a polynomial fit of the resistance:

\[ T_{KTY84} = R^3 \times 0.000000017537 - R^2 \times 0.000109478097f + R \times 0.335831536214 - 143.3 \]
\[ T_{KTY83} = R^3 \times 0.000000010573 - R^2 \times 0.00071592661f + R \times 0.246085727592 - 160.2 \]

5. Conversion of 1000Ω RTD sensors

The resistance of the sensor is computed as resistance = R = ((3.3 × Raw) ÷ tempcurrent) - Offset × 1000. Gain and Offset are typically 1.0 and 0.0 respectively; and tempcurrent is a factory set constant that specifies the current in Amps from the temperature sensor driver (typically 0.00116). Notice that Offset is treated as a resistance offset in kΩ.

The temperature coefficient of resistance (TCR) is computed as TCR = 0.00385 ÷ Gain; where 0.00385 is the nominal TCR of platinum RTDs. The temperature is computed based on the resistance at 0°C (1000Ω), the TCR, and a Callendar second order coefficient of 1.5:

\[ a = 1000 \times (-1.5 \times TCR \div 10000) \]
\[ b = 1000 \times (TCR + TCR \times 1.5 \div 100) \]
\[ c = 1000 \times R \]
\[ T_{RTD} = (-b + (b^2 - 4 \times a \times c)^{0.5}) \div (2 \times a) \]

6. Conversion of sealed gauge sensors

The fuel pressure sensor can be configured to be a sealed-gauge type pressure sensor. To convert this to a true gauge reading the sensor output from step 1 is summed with the difference between standard day pressure (101.325 kPa) and the barometric pressure:

\[ FuelP_{GAUGE} = FuelP_{SEALEDGAUGE} + 101.325 - Baro \]
7. Conversion of CPU temperature

The CPU temperature has its own internal calibration. Gain and Offset are typically 1.0 and 0.0 respectively, however they can be used to adjust the internal calibration.

8. Conversion of digital barometer and OAT

The digital barometer and OAT sensor come from the same integrated circuit: the MS5637. They are sampled much slower than 1kHz; and use a complex conversion process that is documented in the data sheet for the MS5637. Gain and Offset are typically 1.0 and 0.0 respectively; however you can use Gain and Offset to adjust the output if needed.

9. Sensor telemetry

![Image of sensor telemetry](image)

The readings from the sensors are part of the telemetry and displayed on the sensors page. If a sensor is disabled the telemetry is greyed out. If a sensor is invalid the telemetry is red. If you select a failed sensor on the drop down the cause of the failure (reading low or high) is visible on the left of the page.

10. Autocorrect

The MAP, Total Current, and 12V Current sensors all support automatic correction. If enabled the autocorrect feature adjusts the sensor offset to correct known errors in the sensor readings. Autocorrect for the MAP sensor adjusts the offset to make the manifold pressure match the barometric pressure, when the engine is not running. Autocorrect for the current sensors adjusts the offsets to make the current read 0, if the engine is not running, IntelliJect is disabled, and the throttle is more than 10% and less than 90% (i.e. the throttle servo is off the stops).
J. Comms

IntelliJect has three communications interfaces: Universal Serial Bus (USB), RS-232 Universal Asynchronous Receiver Transmitter (UART, i.e. serial), and Controller Area network (CAN). All three interfaces are simultaneously active. A message sent to one interface will (typically) elicit a response on all interfaces. Details of the native communications protocol can be found in the communications ICD, which can be accessed from the display software, see section Error! Reference source not found..

1. Set All to Default

Check this option to set all communications configurations to default values. Once checked you will not be able to change any other options on this window. Default communications run the UART at 57600 bits per second, and the CAN interface at 1Mbit. The CAN interface will be configured for native protocol, using long identifiers (29-bit), with the lower 16 bits of the identifier equal to the serial number.

2. Summary Packets

Check this option to group all the telemetry into a single packet (one packet for fast telemetry, and one packet for slow telemetry), reducing bandwidth. The summary packet option is on by default. This option does not apply to the CAN interface, unless the CAN interface has Packet-Over-CAN selected.

3. Telemetry rates

IntelliJect has two types of telemetry outputs: fast and slow. Fast telemetry is suitable for output at high rates, up to 200Hz, and contains only fast changing information (such as throttle and RPM). Slow telemetry is intended to be output at a reduced rate, and contains a much larger set of
information, including all the sensor data. The default fast and slow telemetry rates are 20 Hz and 1 Hz respectively.

- **Fast rate** is the frequency at which fast telemetry will be sent.
- **Slow rate** is the frequency at which slow telemetry will be sent.

If you select a baud rate (UART or CAN) which is too low to accommodate the rate of fast and slow telemetry the rates will be adjusted down until the expected bandwidth fits the available data rate. The reduction first affects the slow telemetry rate, until that rate is 1Hz, then it reduces the fast telemetry, down to 1Hz if necessary.

4. **UART configuration**

**Bit rate** is used to select the UART data rate in bits per second. The default is 57600. If the display software is connected via serial interface, changing the UART baud rate will cause the display software to wait 100 milliseconds before changing its own serial interface baud rate to match the new baud rate.

5. **SD Card Recording**

SD card recordings are built from normal communications packet; accordingly, SD card options are configured with the communications.

- **Disable** disables all SD card recording, whether the engine is running or not.
- **Always** causes data to be written to the SD card all the time, even if the engine is not running.
- **Fast rate** is the frequency at which fast telemetry is written to the SD card.
- **Slow rate** is the frequency at which slow telemetry is written to the SD card.

6. **CAN configuration**

The controller area network (CAN) transmits data in frames, and each frame has an identifier. In the native protocol a CAN frame identifier is formed by combining a base identifier with the packet type. The combination is done by left shifting the packet type, and OR-ing with the base ID. The base ID for input and output are typically different; and if multiple EFI systems share the same CAN bus they must have different base IDs. In addition, the base ID should have zeros in the six bits occupied by the packet type, see the communications ICD for more information.

If the display software is connected via CAN interface, changing the CAN configuration will cause the display software to wait 100 milliseconds before changing its own CAN interface to match the new settings.

- **Bit rate** is used to select the CAN data rate in bits per second. The default is 1 million bits per second.
- **No Configuration** is used to indicate that configuration change and request packets will not be supported on the CAN bus. Only command and telemetry packets will be supported. All command and telemetry packets are small enough to fit in a single CAN frame. Note that once you have set this option you cannot clear it (or change any configuration parameter) unless you connect via USB or UART.
• **Long ID** should be selected to use CAN protocol 2.0B, which allocates 29-bits to the CAN identifier. If **Long ID** is not selected the CAN protocol is 2.0A, which allocates 11-bits to the CAN identifier.

• **Shift** is the number of bits to left shift the packet type when OR-ing with the base ID. **Shift** cannot be more than 23 bits for long identifiers or more than 5 bits for short identifiers.

• **ID in** gives the base CAN identifier used for native protocol input CAN messages.

• **Disable in** disables the reception of native protocol CAN messages with the **ID in** identifier.

• **ID out** is the base CAN identifier used for native protocol output CAN messages.

• **Disable out** disables the transmission of native protocol CAN messages.

• **ID brd** gives a second base CAN identifier used for native protocol input CAN frames. This is typically used as a broadcast identifier that can be common to all IntelliJects sharing the bus (so throttle or RPM commands can be sent to all devices with a single messages).

• **Disable brd** disables the reception of native protocol CAN messages with the **ID brd** identifier.

• **Packet-over-CAN** forces Intelliject to send all native protocol CAN packets using the packet-over-CAN schema (see the communications ICD for details). Ordinarily packet-over-CAN is not used unless the packet contains more than 8 data bytes. Selecting this option will make the CAN bus less efficient, but it will reduce the number of identifiers that Intelliject uses on the bus (one or two identifiers, depending on if **ID in** is different from **ID out**).

7. **Alternative protocols**

The protocol documented by the communications ICD is the *native* protocol. This is the protocol that the display software implements, and is the normal method to configure, command, and monitor Intelliject. In addition to the native protocol several alternative protocols are supported. If an alternative protocol (except NW GCU CAN) is enabled Intelliject will not output native protocol packets until it first receives a native protocol packet.

• **Address** gives the 16-bit address used with the alternative protocols (except for the PE3 protocol). Typically, this address should be the same as the serial number, but it can be set to any value that is unique on the CAN bus.

• **Currawong UART** enables the protocol used by the [CE367 ECU from Currawong Engineering](https://www.power4flight.com) on the UART interface. Intelliject implements the entire protocol including telemetry, commands, and configuration packets; see the Currawong ECU ICD for details. When this protocol is enabled the UART bit rate is forced to 57600 bits/s.

• **Currawong CAN** is the same as **Currawong UART**, but specifically for the CAN bus. It is possible to have the currawong protocol enabled on both the CAN and UART interface simultaneously. When this protocol is enabled the CAN bit rate is forced to 1M bit/s.

• **Piccolo CAN** enables an alternative protocol that is supported by the [Piccolo autopilot](https://www.power4flight.com). It is only available on the CAN bus, and it cannot be enabled simultaneously with **Currawong CAN**. Selecting **Currawong CAN** will force the CAN bit rate to 1M bit/s.

• **PE3 CAN** enables the performance electronics output-only CAN bus protocol used on the PE3 ECU. This protocol can be simultaneously enabled with the **Piccolo CAN** protocol.

• **NW GCU CAN** enables the reception of telemetry from the Northwest Generator Control Unit (GCU). This is input-only data. Reception of this telemetry will cause Intelliject to generate the native protocol GCU telemetry message.
8. Packet counters

IntelliJect maintains packet counters, which roll over after 255.

- **CAN tx** counts the number of native packets that were sent on CAN.
- **Tx cnt** counts the number of native packets sent over USB or UART.
- **Rx cnt** counts the number of native packets received from any interface.

9. CAN errors

The CAN specification includes a detailed error mechanism that dictates how a CAN device detects and handles errors on the bus.

- **REC** is the receive error counter, which increments each time a CAN message is incorrectly received. It counts down anytime a CAN message is correctly received. The **REC** should be 0 if IntelliJect is operating normally on a CAN bus.
- **TEC** is the transmit error counter, which increments each time a transmitted CAN message sent was not acknowledged by the bus, and decrements each time a transmitted message was acknowledged. The **TEC** should be 0 if IntelliJect is operating normally on a CAN bus.
- The most recent CAN bus error is given below the **TEC**. This will be one of: “No error”, “Stuff”, “Form”, “Acknowledgement”, “Bit recessive”, “Bit dominant”, “CRC”, or “Software”.
- **Bus off** is set if the error counters have reached the bus off limit.
- **Error passive** is set if the error counters have reached the passive error limit.
- **Warning limit** is set if the error counters have reached the warning limit.

In addition to the errors given by the CAN specification, IntelliJect maintains two other error flags for the CAN bus:

- **Tx buffer full** is set if IntelliJect was unable to transmit all its data on the CAN bus during the last slow telemetry epoch. This will happen anytime the CAN bus is disconnected or is overly congested.
- **Rx failures** is a counter that indicates how many native protocol packets received over the CAN bus were invalid during the last slow telemetry epoch. Since small packets (8 or fewer data bytes) are handled directly by CAN messages this failure counter only applies to large packets sent using packet-over-CAN.

10. USB and UART errors

The USB and UART interfaces each maintain these two error flags.

- **Tx buffer full** is set if IntelliJect was unable to transmit all its data on the interface during the last slow telemetry epoch.
- **Rx failures** is a counter that indicates how many native protocol packets received over the interface were invalid (i.e. had a CRC failure) during the last slow telemetry epoch.
K. Locking

Locking controls locking and hiding of configuration data. This page also contains other commands related to configuration storage. The configuration is locked by default. Any attempt to change configuration will cause IntelliJect to ignore the change and issue a not-acknowledge (nack) message (see the communications ICD for more details).

Configuration locking is a safety mechanism to prevent unintended changes. You do not need to unlock the configuration to run an engine, only to change configuration. To change configuration, you first unlock the configuration using the buttons at the top of the configuration window.

Figure 32: Locking window, with the configuration unlocked.

Figure 33: Configuration locked, Enter unlock password is visible.

In Figure 33 Enter unlock password is visible. In this case the configuration was locked with a password, and IntelliJect is reporting that it needs a password to be fully unlocked. To unlock: enter the password\(^5\) and hit enter, or the Unlock button. If you do not enter a password, or the password is incorrect, you will get a nack message (Figure 34) and IntelliJect will enter the partial

\(^5\) The password is encoded in plain text in the unlock message. Anyone who can intercept your communications will be able to read the password. Since the interface connection is typically direct, this is unlikely; and cryptographic quality security is not warranted. Note that IntelliJect will never send the password.
**unlock** state (Figure 35). There are three lock states: locked, partial unlock, and fully unlocked (the state in Figure 32).

![Nack](image)

**Figure 34:** Not-acknowledge message.

![IntelliJect](image)

**Figure 35:** IntelliJect in partial unlock state.

1. Configuration data access when partially unlocked

When locked, data cannot be changed; when fully unlocked, all data can be changed. The ambiguous state is **partial unlock**. For each category of configuration data, the data access controls what to do when partially unlocked.

- **Read/Write**: indicates the data are changeable. This means that a user does not need the password to change this data, since even partial unlock provides adequate permissions.
- **Read Only**: indicates that data are not changeable, changes will be rejected with a nack.
- **Hidden**: indicates the data are not changeable and cannot be requested, IntelliJect will nack a request for that data. The request prohibition also applies to the locked state.

If IntelliJect is not locked with a password the **Enter unlock password** field will not be visible. If a category is hidden the display software will prevent you from viewing that page of data. In Figure 35 you can see that the **Tables** and **Sensors** pages are not active, as those categories were configured to be hidden when partially unlocked. The display software is smart enough to prevent access to this data, however even if you use some other software you will simply get a nack message if you request the hidden data.

To change the access controls, you need to be fully unlocked. Then you can select the **Read/Write**, **Read Only**, or **Hidden** choices as needed. Note that you must use the **Change Access** button to send the locking configuration message, the **Send** button does not send the message. If you want to lock with a password enter it in the **Lock password** field before hitting **Change Access**. Just leave the password blank if you do not want a password.

Since you cannot change the locking configuration without the password, recovering from a lost password is impossible. To deal with this you can use the hard-coded password “CLEAR ALL DATA”

---

---

6 By switching between offline and online it is possible to fool the display software into showing you pages that should be hidden. Note however that IntelliJect will still not send the data if requested, so any information visible on such a page will not be valid.
(without the quotes, case sensitive). This will unlock, but it will also clear all stored configuration data, both user and factory storage data, returning IntelliJect to its default configuration. If the engine is running when you issue this command it will be ignored and a “engine is running” nack will be returned instead, see Figure 36. You will have to stop the engine to proceed.

2. Configuration file and comment

Use the **Change Comment** button to apply or change the configuration comment. There is also a file and date field in this part of the window. If you use the menu **File->Open All...** to load a configuration the file name and the current date can be sent to IntelliJect and are visible here. This is a handy way to know what file was used for configuration. Note that a configuration file can include a comment.

3. Settings storage

There are two forms of configuration storage: user storage and factory storage. When IntelliJect boots up it defaults all configuration data. Then it loads configuration data from the factory storage, potentially overriding default configuration data. Then it loads configuration data from the user storage, potentially overriding factory data.

Any time you change configuration you are changing the user storage. These settings are stored as soon as they are received, even if the engine is running. The storage operation is performed on a per-packet basis and a journaling system is used to prevent data corruption if the power is lost during the store operation.

Factory storage uses on-chip flash (the same nonvolatile space that holds the firmware). This storage cannot be written in a granular (per-packet) fashion; and it cannot be written while other code is executing, which means the engine must stop running to write the factory storage. The only way to write factory storage is to use the button **Save to Factory**; which causes all the configuration data to be written to the factory storage and cleared from the user storage. The only exception is the engine wear information, which is maintained in user storage (since it is always changing).

The **Restore Factory** button discards all user storage data (except engine wear data) and reloads the configuration from factory storage. This is a convenient way to return to a previously known good configuration. The **Clear All** button discards all storage (user, factory, and logbook; including wear data), reverting to the default configuration. Since **Restore Factory** and **Clear All** both change a lot of configuration data they will trigger a send of all configuration data from IntelliJect to the user.

If the engine is running when the **Save to Factory** or **Clear All** operation is commanded the command will be ignored and you will get a nack message, see Figure 36.
4. Settings mismatch check

Once you have arrived at a final configuration you should use the button **Set Configuration Valid**. This will command IntelliJect to compute a hash of the current configuration and store that hash, finishing by locking. IntelliJect will track the hash of the actual configuration and compare it against the stored value. If a mismatch occurs the parameters mismatch error is asserted, letting you know that something has changed from the last known good configuration.

The hash that was stored is given in the **Settings Hash** display. The hash that was computed from the current configuration is given in the **Actual Hash** display. You can disable this error check (for example during engine calibration when configuration is changing frequently) by using the **Disable Mismatch Check** button.
L. Maintenance Schedule

This section allows you to specify the engine maintenance schedule, which consists of a list of individual maintenance items. Each schedule item includes a description of the maintenance, a trigger type (for example, Engine Time), and an interval for performing the maintenance. Intelliject also maintains status information for each schedule item, see section VIII.B for more details. Figure 37 shows the configuration maintenance window, which looks like the maintenance status display (section VIII.B), but with the controls to make schedule changes.

![Figure 37: Maintenance schedule configuration.](image)

1. Hot Time Gain and Load Threshold
   - **Hot Time Gain** determines how hot time is accumulated in the engine wear. Anytime CHT is above “CHT too hot” (section V.F) hot time is accumulated at a rate determined by the gain multiplied by the temperature excess. For example, if the CHT is 20°C too hot, and the gain is 0.1, hot time will accumulate 2 seconds for every 1 second of real time. Hot time only accumulates when CHT is above the limit; and never accumulates slower than real time.
   - **Load Threshold** determines the throttle position above which Load Time will accumulate in the engine wear. Load Time always accumulates at real time rate.

2. Maintenance triggers

Use the Type drop down to select the event that triggers maintenance. Each of the triggers has a corresponding value in the engine wear data (section VIII.A); and the type of trigger defines the units of the maintenance interval. Available triggers are:

- **Engine time**: total amount of time the engine has been running; interval and last are in hours
- **Revolutions**: total number of engine revolutions; interval and last are in millions of engine revolutions.
- **Hot Time**: amount of pseudo time the engine has run at temperatures above “CHT too hot”. Hot time interval and last are in hours, however it is not a true time, as it is accumulated on an accelerated schedule based on how much temperature excess the engine experiences.
- **Load Time**: amount of time the engine has spent at high loads; interval and last are in hours.
• **Starts:** number of times the engine has been started; *interval* and *last* are in units of starts. To record a start IntelliJect must see 1000 revolutions after the RPM was zero.

• **CHT:** is the cylinder head temperature while the engine was running. Unlike the other maintenance triggers the *interval* does not specify a difference from the last service; instead the maintenance comes due when the head temperature exceeds the *interval* value. The *interval* is given in degrees Celsius, and when *CHT* maintenance is done the *last* value records the engine hours.

3. **Changing the schedule**

To add a maintenance item to the schedule, push the **Add Schedule Item...** button. The software prompts for the item number, allowing you to insert the item anywhere in the schedule list. Removing a maintenance item is done with the **Remove Schedule Item...** button, which will also prompt for the number of the item to remove. Using these buttons does not actually change the maintenance schedule on IntelliJect; because adding a maintenance item also requires editing the type, interval, and description. After adding, removing, or editing a maintenance item you push the **Send Schedule** button to send the entire schedule to IntelliJect. **Send Schedule** is red anytime the schedule is edited to indicate that it needs to be sent.

IntelliJect compares new schedule to the previous schedule. Any item in the new schedule which matches the *type* and *description* from an item in the old schedule will preserve that item’s maintenance status; even if the item has changed position in the list. If you change type, or description, or if you remove a maintenance item, the corresponding maintenance status is lost.
M. Configuration files

Configuration files store the same information that IntelliJect keeps in its non-volatile storage. You use configuration files for configuration management, and to move configurations from one IntelliJect to another.

1. Saving configuration files

When using the file menu **Save IntelliJect Config...** the software will ask what information should be saved. Each check in Figure 38 represents a single packet’s worth of information that will be written to the file (except for Engine logbook, which could be multiple packets). For convenience the configuration data are separated into categories; and you can turn an entire category on or off using the **All** or **None** buttons.

![Figure 38: Configuration selection save window.](#)

When selecting a file to save you have a choice of fie extension, “.efi” or “.efitext”. If you do not supply an extensions the default choice will be “.efi”. This file stores configuration data as an order-independent series of binary packets, exactly as documented in the communications ICD. The “.efitext” file stores configuration data as a human readable text file with a line of text for every field of configuration data.
If you overwrite an existing configuration file, only the selected data are overwritten, any other configuration data that were previously in the file will be preserved. In this way it is easy to update one configuration file with pieces of information from another file.

2. Opening configuration files

When using the file menu **Open EFI Config**... the software will first check what data are in the file and then ask what should be read from the file. In Figure 39 the file contains a limited set of data. You can deselect any of these items to further winnow what information is read from the file. If IntelliJect is online the display software will send the selected configuration data; otherwise the display will simply be populated with the data loaded from the file. The software automatically determines if the configuration data are stored in binary or text format based on the file extension.

When opening configuration files, you will typically not send maintenance data, since maintenance data are not specific to an engine configuration, but instead specific to an individual engine. Therefore, the maintenance check boxes are clear by default, even if the file contains maintenance data.

![Figure 39: Configuration selection window for opening configuration data.](image)

3. Comparing configurations

The file menu **Compare IntelliJect Config**... opens the configuration comparison window. The configuration comparison allows you to compare two configuration files, or a single file can be compared against the configuration of the IntelliJect.
• **Select first file...** is used to select the first file to compare.
• **Use IntelliJect data** should be selected to compare the first file against IntelliJect, rather than a second file.
• **Select second file...** is used to select the second file to compare.

Once the file(s) are selected the comparison window asks what parts of the configuration should be compared, using a window like Figure 39. Each field of the selected configuration is compared, and the differences are reported as text in the comparison window.

In Figure 41 the configuration file “testing.efi” was compared against IntelliJect and configuration differences were found in the throttle, the main fuel table, and the temp difference table. If you want to save the difference report you can copy the text directly from the window.
VI. HOW TO CONFIGURE AND CALIBRATE INTELLJECT

Configuring IntelliJect to properly run an engine is an involved and complex process best undertaken by engineers experienced in fuel injection with access to an appropriately sized dynamometer and sophisticated measurement equipment. This section provides the rough order of operations to use in configuring and calibrating IntelliJect. You should already be intimately familiar with the configuration section of this manual (section IV). If you have no a-priori configuration data it is best to start in the default configuration (i.e. the condition after commanding clear all data, section V.K.3).

A. Hardware setup

1. Throttle

Configure your throttle hardware, whether you will use a TPS sensor, or directly drive the throttle with a servo (or both). Most IntelliJect applications use an alpha-n control strategy without real time feedback of the engine mixture. Therefore, the throttle calibration must be done as exactly (and repeatably) as possible. Make sure this is correct now, fixing an incorrect throttle calibration later will invalidate much of the engine calibration.

During calibration it is recommended that you disable the throttle curve. This can be easily added after calibration. If IntelliJect is driving the throttle it is recommended that you do not use the analog or PWM throttle command method, instead use the digital throttle command for more exact throttle control.

2. Engine

Make sure you have the displacement set correctly (if you get this wrong you can fix it later by multiplying the fuel table). Set the CHT reference to the temperature you expect to use during the determination of the fuel table.

Set up the crank sense settings. This should be done with care, particularly the measurement of the crank angle at which the sense synchronizes. If you get this angle incorrect you will need to adjust the spark table later when you fix it. If you change the crank sense hardware you should be able to update the crank sense settings without changing any other part of the configuration.

Setup the ignition configuration. If you will be using a dual plug ignition, decide now if you want to delay spark 2 or 3 from spark 1 and set the spark 2 delay table and spark 3 delay table (section V.B.7) appropriately.

Setup the cooling and desired engine head temperatures. Give some thought to this now. Although IntelliJect should operate the engine well at a wide range of head temperatures you’ll get the best result if the engine operates near the same head temperature used during calibration.

Set the soft and hard rev limits.
3. Injectors

Configure your injector hardware, most importantly the injector calibration. Get this as precise as you can. Be wary of measuring fuel flow with an injector spraying into a beaker on a scale. Gasoline mist is extremely flammable and so fine that substantial amounts will vaporize into the air, invalidating the weight measurement. If you use this method, cover the beaker and injector with plastic film to constrain the fuel vapors. A better tool for measuring fuel flow rate is the Max Machinery flow meter (section XXVII.D.3). Use a hydrometer to get an accurate fuel density measurement; and use test mode to get precise control of the injector output.

![Injector Type: example injector](image)

Measuring the flow rate at different pulse widths can generate a linear regression which will give you the injector flow rate and the trim. You can also determine the narrowest pulse that still follows the linear regression. A procedure for doing this using IntelliJect test mode (XXII.A.3) would work like this:

- Setup test mode for 3000 RPM (50Hz) and 60 second test; for injector 1.
- Set the duty cycle to 5%, this will result in an injector pulse of 1 millisecond (0.05/50).
- Zero the fuel delivered on the flow meter (section XXVII.D.3).
- Execute the test. IntelliJect will run the test for precisely 60 seconds.
• Record the amount of fuel delivered, re-zero the fuel delivered measurement.
• Change the duty cycle to the next test case (5%, 10%, 20%, 40%, 60%, 80%).
• Rerun the test, repeating until 80% duty cycle or your fuel pump cannot keep up.

Running the above procedure on an example injector will produce results like Figure 42. Notice that the regression used to determine the slope and offset of the injector characterization must be manually updated to exclude injector openings which are too short to lie in the linear operating region.

You should make these measurements at different fuel pressures. The injector trim measurements go in the Injector trim tables. The flow rate and minimum pulse measurements go into the injector configuration category. If you are using more than one injector be sure to calibrate both and set the injector ratio table appropriately.

Choose if you want the injector phasing to control the start, middle, or end of the injection phase. If you don’t know which is best for your application, choose middle phasing. Set the output multiplier and fuel used multiplier to 1.0.

4. Fuel pump

Setup the fuel pump control, either bang-bang or proportional. Make sure to set the fuel pressure based on your requirements for the system, 300 kPa is commonly used for manifold injection setups. If IntelliJect is not controlling the fuel pump you still want to set the pressure error limits. Note that you can change your mind later. IntelliJect will automatically correct the injectors for fuel pressure, so you can change pressure in the future.

IntelliJect is often used with single piston positive displacement fuel pumps from Currawong Engineering. If this is your setup you will want bang-bang control, and you will need to include a fuel pressure accumulator to soften the pressure spikes. The bang-bang hysteresis can be set to your preference but there is little value in trying to make it smaller than 10 kPa.

5. Sensors

Setup the sensor calibrations, this is mostly about the temperature sensors. However, if you use an analog TPS you will calibrate that here. Make sure you correctly set whether your fuel pressure sensor is sealed-gauge or not. If you are at standard day conditions it won’t matter, but as you deviate pressure from 101.325 kPa the sealed-gauge setting becomes important.

Pay attention to the default values for the sensors. These are the numbers that IntelliJect will use if a sensor fails and there is no other backup.

The remainder of the configuration settings can be set to your preference, but you should probably:

• Make sure your fast and slow telemetry data rates are as high as you want them.
• Turn off settings mismatch detection and clear the locking password. These will be a nuisance until you’ve finished the calibration.
• Set a comment that reflects the status of the calibration.
• Save your settings to a file so you can easily go back.
B. Calibration

Calibration is ideally done on a motoring dynamometer that will drive the engine to a specific speed, no matter what torque the engine produces. The order of operations here assumes you have such a tool; if you don’t, do your best, but be warned that engine calibrations which are not done on a dyno are likely to be poor.

1. Initial setup

The default fuel table contains some numbers, which are likely to be a long way from good but can get you started. Setup the rows and columns you want for the table. You can change these later by adding or deleting rows or columns, so it’s your choice regarding how much density to start with. When you insert rows or columns software will interpolate against any data you currently have in the table.

Before getting started check the spark table. A reasonable safe starting point is 15 degrees of spark advance. You cannot optimize the spark table until the fuel table is roughed in, so just make sure that nothing likely to break the engine is in the spark table.

2. Head temperature fuel compensation

If you have good enough cooling control to maintain head temperature during the calibration, and if you set the CHT reference value, then you can delay the head temperature compensation until later. Otherwise, you will need to determine this temperature dependence now, before going further with the fuel calibration. If you don’t have the compensation appropriately determined first, the fuel values you choose will be limited to the head temperatures at which you calibrated.

Using the dyno get the engine operating at an initial speed and throttle (say, a typical fast idle condition). You should be at a speed and throttle that exactly matches a cell in the fuel table. Monitor and adjust the cooling system to get the engine to the intended operating temperature. Adjust the value in the fuel table until the torque measured by the dyno is at its maximum.

Now use the cooling system to vary the engine head temperature, going up or down as you see fit. If you need to adjust the fuel value to keep the engine at maximum torque (or to maintain mixture as indicated by a gas analyzer) then you need to add information to either the charge temperature table (section V.B.9) or the cylinder head temperature table (section V.B.10). Which table you use is a personal choice. Iterate the head temperature compensation until changing the head temperature does not change the mixture.

Do the same set of steps for other combinations of load and speed. Use as much table density as you need to capture the temperature dependence trends of the engine. A typical head temperature compensation for a single cylinder two stroke might include three loads and three RPMs.

---

7 It depends on what you believe is happening when the temperature changes. I prefer the charge temperature table, as it is a closer model of inlet tract heating.
3. Injector phase

If you expect injector phase (crankshaft angle when the injection happens) may impact performance, now is the time to explore that. A typical manifold injected single cylinder two stroke will have very little dependence on injector phasing, but other engine topologies may be more significantly impacted.

For example, manifold-injected common-crank-case boxers will show cylinder balance variation with injector phasing. You will want to adjust the injector phase either to maximize power, maximize efficiency, or balance the mixture across cylinders. Detailed injector phasing studies will require a high-quality dyno, fuel flow meter, and exhaust gas analyzer.

It is common to adjust injector phase simultaneously with head temperature compensation. The two tables typically need a similar amount of density and using the same operating points for both is reasonable.

4. Initial spark calibration

At this point you should have a range of loads and speeds with reasonable fuel values. Now is a good time to set the spark advance for those same conditions. Start at a low advance and increase (within reason) to reach maximum torque. Avoid choosing the absolute maximum torque, typically you can back off the maximum torque by a few degrees of advance without sacrificing much performance.

Use the values learned here to make guesses for the rest of the spark table (or add rows and columns and let the software interpolate). Getting the spark roughed in is useful since the spark timing may have some modest impact on the volumetric efficiency, which will affect the fuel value.

5. Complete fuel calibration

Go through every cell in the fuel table and adjust the fuel value to get the desired performance. A good starting point is to choose the fuel that maximizes torque. However, you can get better efficiency by using less fuel. Do not go leaner than the max efficiency point; in fact, stay well rich of that point, most engines will not run there unless they have a very good combustion chamber and lots of flywheel mass. A reasonable strategy is to target maximum torque at high and low throttle, and a more efficient setting in the cruise range.

You can test if you have enough density in your fuel table by going to an operating point that is mid-way between cells in the table and checking the engine performance. If the engine is rich or lean you should probably add a row or column there, and tailor the setting for that point.

6. Complete spark calibration

Typically, you will want the spark table to have the same loads and speeds as the fuel table, so set that now (the table clone feature makes that easy). Adjust the spark advance at every cell as you did before when roughing out the spark calibration (section VI.B.4).

This is the last setting you will make which affects the nominal engine performance. Therefore, after each cell has been adjusted do the following:
• Record any data you need to capture the engine performance. Typically: power, fuel flow, and exhaust gas data.
• Add an entry to the MAP estimate table that gives the ratio of manifold to barometric pressure at that operating point.
• Add an entry to the shaft power estimate table that gives the corrected-to-standard-day power at that operating point.

C. Polishing the calibration

1. Drivability

Connect the engine to its normal load (prop shaft, generator, etc) and run it to determine drivability. Dynos have a lot of inertia, and the change in flywheel mass from the dyno to the actual load can raise issues. Typically, you will need to reduce the spark advance at low engine speed to get smoother operation. You might also tweak the fuel and spark to optimize the transitions.

Idling can often be improved by further detuning the low speed spark advance. For example, the desired idle speed might be achieved at 5% throttle with 20 degrees spark advance, or at 10% throttle with 5 degrees advance. Although it is less efficient the latter case is likely to be better because of the larger throttle opening, which will be more repeatable unit to unit. It is even possible to add a bit of stall resistance by detuning the spark at the desired idle speed, but not at lower speeds, hence if the engine stumbles it quickly picks back up.

This is also the time to input a throttle curve, if IntelliJect is driving the throttle. A curve goes a long way to making an engine behave tractably. The best curve to use depends on the load; however, if the load is a fixed pitch propeller a good choice is to adjust the curve so RPM cubed is proportional to throttle command (propellers absorb power in proportion to the cube of RPM under static conditions). The curve builder feature (section V.C.5) can be used for this.

2. Starting

Adjust the starting fuel table (section V.B.15) to get the quickest and most reliable engine start you can at cold temperatures. This table has default values that are better than nothing, but you can typically make it much better. You will need a way to cold soak the engine and fuel to experiment with this.

Set an appropriate value for the maximum starting throttle (section V.C.3). By preventing accidentally starting an engine at high throttle you improve both safety and engine longevity.

3. Warm up

The cylinder head temperature table (section V.B.10) can be used to add fuel for a cold engine. For warm up temperatures (typically less than 80C) you will typically need more fuel than the compensation you determined during the calibration above. Add rows to the cylinder head temperature table as needed; remember IntelliJect will not extrapolate beyond the end of the table, so include the coldest head temperature you think you’ll see.
Warm up fuel is different than starting fuel. Starting fuel is for quickly wetting the inside of the engine with fuel, and it cuts off after the specified number of revolutions. Warm up fuel deals with poor fuel vaporization at low temperatures; and remains in effect while the engine is cold.

If IntelliJect is driving the throttle the minimum throttle table (section V.B.21) can be used to specify the lowest allowed throttle, as a function of head temperature. Cold engines do not idle as well as warm engines; use this table to hold the throttle open when the engine is cold, preventing rough running or stalling. Similarly, you can use the maximum throttle table (section V.B.23) to prevent damaging the engine by opening the throttle too far when the engine is cold.

4. Over heating

If cooling is at maximum and the engine is still overheating the next best option is to reduce the power output by limiting the throttle; use the maximum throttle table (section V.B.23) for this.

If the engine is prone to knocking you should use the spark retard table (V.B.8) to reduce the spark advance when the engine is hot. You can also use the cylinder head temperature table (V.B.10) to add more fuel when the engine is overheating. The additional fuel will provide a small improvement in cooling, and it will reduce the likelihood of knock.

5. Maintenance

Take the time to define a maintenance schedule (section V.L). Customers appreciate having some guidance on engine service.

6. Final steps

- Tune the RPM controller if you are going to use that feature (section V.D).
- Adjust your comment to reflect the finished state of your calibration (section V.K.2).
- Set the locking configuration and password appropriately (section V.K.1).
- Set your configuration valid (section V.K.4).
- Save the configuration to a file (section V.M.1).
- Consider storing the configuration to factory settings (section V.K.3).
VII. ENGINE LOGBOOK

The engine logbook is a digital record used to track engine status. The logbook consists of a series of entries, each with a date, name (20 characters max), and description (995 characters max). The logbook can only be edited when the engine is not running. If IntelliJect is locked the only logbook action allowed is to add new entries with the New... button. If IntelliJect is unlocked entries can also be edited and deleted, or the entire logbook can be erased.

Figure 43: Logbook window with context menu open

A. Logbook files

The logbook can be saved or loaded from a file in the same way as other IntelliJect configuration data, see section V.M for more details. Or you can use the Save... and Open... buttons for a more human friendly file format. When you open a log file the entries contained within are added to the logbook onboard the EFI. If you want to completely replace the logbook you must first Erase... the logbook on IntelliJect.

Logbook entries can be deleted or edited by right-clicking to invoke the context menu and selecting Delete entry... or Edit entry.... When adding or editing a logbook entry you fill out the dialog in Figure 44.

As an alternative to saving the logbook to a file you can copy specific entries to the clipboard, using the copy shortcut or the Copy command on the context menu. The format of the copied data is identical to the format of the log entries saved to a file.

B. Logbook storage details

The logbook is stored onboard IntelliJect in 128Kbytes of dedicated flash. If each entry uses the maximum text there is space for at least 127 entries. Most entries will be smaller, so it should be
possible to have many more entries, up to a maximum of 1022. Because the logbook is stored in flash deleting or changing an entry requires zero-ing that flash location and re-writing the entry further on. Too many entries, deletions, or edits, will exhaust the logbook storage space. In that case logbook additions will not be accepted, and the logbook must be erased and reloaded from a file. Anytime a logbook entry is received IntelliJect will only update the logbook if the new entry is unique; this eliminates the amount of logbook space that would otherwise be used when resending logbook entries that are already onboard IntelliJect.
VIII. ENGINE WEAR AND MAINTENANCE

A. Engine Wear

As part of its regular telemetry IntelliJect reports engine wear information.

- *Time* is the number of hours the engine has been run.
- *Revs* is millions of engine revolutions.
- *Peak CHT* is the hottest CHT measured while the engine was running, in degrees Celsius.
- *Hot Time* is amount of run time at temperatures above “CHT too hot” (see section V.F). *Hot time* is accumulated on an accelerated schedule based on the temperature excess, see section V.L for more details.
- *Load Time* is the time at high loads, see section V.L for the definition of high load.
- *Starts*: is the number of times the engine has been started. To record a start IntelliJect must see 1000 revolutions after the RPM was zero.

![Figure 45: Engine wear and maintenance status.](image)

Normally engine wear is not changed by the user, but if IntelliJect is moved to another engine the wear information and serial number may need to be changed. With IntelliJect unlocked push the *Change Wear*... button to get the dialog in Figure 46.

![Figure 46: Engine wear input.](image)

B. Maintenance status

IntelliJect includes a configurable maintenance schedule, see section V.L. The status of each maintenance item is visible in Figure 45. The status indicates if maintenance is needed, when the
last time the maintenance was done, and the name of the person who performed the maintenance.

To determine if maintenance is needed IntelliJect compares the engine wear information against the maintenance schedule and status. If maintenance is needed the maintenance bit is set in the errors telemetry, see section VI. In Figure 45 maintenance items 2, 5, and 7 have been done in the past, and maintenance 6 needs to be done.

1. Performing maintenance

To tell IntelliJect that maintenance has been performed push the Do button under the status column (the button will be OK if maintenance is not due). The software will ask for the same information as a log entry, with the text pre-populated based on the description of the maintenance item. This information is sent to IntelliJect.

When IntelliJect receives the information it will update the maintenance status with the corresponding value from engine wear; clearing the error indication. IntelliJect will also add an entry to the engine logbook (see section VI) recording that the maintenance was performed. You can Do maintenance even if a maintenance item is not due for service.
IX. Fault Detection and Handling

IntelliJect has many hardware and software features for detecting and handling faults, enabling sophisticated “limp home” modes of operation to keep an engine running.

A. Faults due to sensor problems

Most of the limp home modes are related to sensor failures. For purposes of fault handling a sensor that is disabled is equivalent to a sensor that is failed. For example, one good CHT sensor and one disabled CHT sensor is functionally equivalent to one good CHT sensor and one failed CHT sensor. The only difference is that disabled sensors do not set error bits in the telemetry.

1. Crank sensor failure

Crank sensor failure can occur through failure of the sensor integrated circuit (IC), or through disconnection in the harness. Sensor failure is detectable if both crank sense1 and 2 are installed and enabled. The crank sense2 event checks the elapsed time since the last crank sense1 event, and vice versa, to determine if a sensor has failed; making it possible to detect a failed crank sensor after one bad revolution. If a crank sensor is failed the spark and injector outputs will be scheduled by the other sensor.

Crank sensors may have intermittent errors, and IntelliJect records the time of the last error. IntelliJect will use the sensor whose error is oldest in time, until the error is more than 1 minute old. In this way a “glitchy” or intermittent sensor will not be used; even if it momentarily reads as good. Only if both sensors are healthy for more than 1 minute will IntelliJect follow the normal crank sensor preference described in the engine configuration (section V.E.3).

2. Cylinder head temperature sensor failure

Failure of a CHT is detected using under-range and over-range thresholds. A disconnected CHT produces an over-range measurement, and a shorted CHT produces an under-range measurement. A failed sensor has its measurement replaced with the good sensor measurement. If all the sensors fail, the CHT measurements are the default sensor values. In addition, if all sensors are failed, the cooling output is driven by the feedforward gain of the cooling control law.

It is possible to have a third CHT sensor, if the Spare temp sensor is enabled and if the injector configuration specifies that the Spare temp sensor is used for CHT3. In that case if the Spare temp sensor is healthy its measurement participates in the CHT failure logic.

3. Manifold temperature sensor failure

Failure of MAT is detected using under-range and over-range thresholds. A disconnected MAT produces an over-range measurement, and a shorted MAT produces an under-range measurement. It is possible to have a second MAT sensor, if the Spare temp sensor is enabled and if the injector configuration specifies that the Spare temp sensor is used for MAT2. In that case if the Spare temp sensor is healthy its measurement will replace the MAT sensor measurement if it is failed.
If all MAT sensors fail there are two options: if the OAT sensor is enabled and healthy the manifold temperature is taken from the OAT sensor. If the OAT sensor is not healthy MAT is computed from the US standard atmosphere temperature based on the barometric pressure. The MAT default sensor value is used to specify the temperature at standard day pressure (101.325 kPa), and the standard atmosphere lookup will adjust the temperature accordingly.

The OAT sensor is on the IntelliJect board, as part of the IC that provides the digital barometric pressure. Depending on where the board is installed it could be significantly hotter than the ambient air. In some installations it may be better to disable the OAT sensor, thereby selecting standard atmosphere as the MAT limp home.

4. Fuel pressure sensor failure

Failure of fuel pressure is detected using under-range and over-range thresholds. A disconnected fuel pressure sensor produces an under-range measurement. If the fuel pressure sensor fails the sensor reading becomes the default sensor value, and fuel pressure compensation is determined from that value. In addition, the fuel pump output can be configured to go to maximum output (appropriate if a mechanical regulator is included) or the output can be based on the feedforward term, which determines pump duty cycle from the desired fuel flow.

5. Manifold pressure sensor failure

Failure of the MAP sensor is detected using under-range and over-range thresholds. If the sensor fails and the engine is not running, or the MAP estimate table is disabled, the manifold pressure is replaced with the barometric pressure, if it is good, otherwise the manifold pressure is the default sensor value. If the sensor fails and the engine is running, and the MAP estimate table is enabled, the manifold pressure is computed using the table (with throttle and RPM as inputs) and barometric pressure.

6. Barometric pressure sensor failure

Failure of the analog barometric pressure sensor is detected using under-range and over-range thresholds. Failure of the backup digital pressure sensor is detected through failure to read the digital interface. If the analog sensor fails, and the digital sensor is good, the digital sensor value is used. If both sensors fail and the engine is not running, or the MAP estimate table is disabled, the barometric pressure is replaced with the manifold pressure, if it is good, otherwise the barometric pressure is the default sensor value. If the sensor fails and the engine is running, and the MAP estimate table is enabled, the barometric pressure is computed using reverse application of the MAP estimate table.

7. Analog or PWM TPS sensor failure

Failure of the analog TPS sensor is detected using under-range and over-range thresholds. A disconnected analog TPS sensor produces an under-range output. PWM TPS sensor failure is detected when the elapsed time from the last PWM edge is too long. If the sensor fails, and if IntelliJect is configured to drive the throttle, the TPS is assumed to be the throttle output.

If IntelliJect is not configured to drive the throttle, and the engine is running, and the manifold pressure is good, the TPS is computed through reverse lookup of the MAP estimate table. If
IntelliJect is not configured to drive the throttle, and the engine is not running, the TPS becomes the starting throttle value from the throttle configuration.

8. Excess or Insufficient manifold pressure

If the MAP estimate table is enabled, and the barometric pressure is good, and the engine is running, the measured manifold pressure is compared against the predicted value from the table. If the difference is more than the maximum manifold pressure difference from the engine configuration the manifold pressure is replaced with the results from the table.

If the engine is not running, and the barometric pressure is good, the manifold pressure is compared against the barometric pressure, and the difference between the two sensors is removed by adjusting the MAP sensor offset. This corrects for long term drift of the MAP sensor caused by fuel exposure.

B. Faults due to mechanical, environmental, or software problems

1. Excess or Insufficient cooling

If the cooling system cannot keep the engine cool enough, and if IntelliJect is configured to drive the throttle, the maximum throttle table can be configured to limit the engine power to prevent catastrophic overheating. If the engine is too cold, and if IntelliJect is configured to drive the throttle, the minimum throttle table can be configured to keep the engine power high enough to prevent engine stall.

2. Excess or insufficient fuel pressure

The measured fuel pressure is used to adjust the injector opening time to account for any deviation from the desired fuel pressure. The injector time is adjusted according to the square root of the ratio of the nominal injector pressure to the measured pressure. This allows the fuel pressure to vary substantially from the desired value with little impact to the engine performance or fuel flow.

3. Ignition failure

Failure of an ignition can occur due to failure of the ignition hardware, the spark plug, the spark plug cap, or the harness. IntelliJect has two ignition outputs, which can be independently scheduled and enabled. When paired with engines that have dual spark plugs this allows the engine to continue running if one of the ignitions fails. IntelliJect also has a feature to individually disable an ignition output, allowing you to test the effects of a failed ignition; and allowing you to verify both ignitions are operating prior to launch.

4. Power fault

IntelliJect supports a special bootup mode referred to as quick restart. If the engine is running IntelliJect saves key engine control variables to backup RAM. The contents of backup RAM are preserved for several seconds following loss of power.
The normal bootup process for IntelliJect will take too long to keep an engine running; since the normal process performs a code CRC check and reads configuration data from EEPROM. If quick restart is active the CRC check is skipped, and the crank sense, spark, and injector hardware are setup using the quick restart data while waiting for the configuration data read to finish. When quick restart is active It takes IntelliJect 40 milliseconds to reboot and initialize to the point where it can schedule spark and injection. If the power briefly falls below the minimum input voltage IntelliJect can reboot fast enough to keep the engine running.

5. Software fault

Every fault and interrupt the processor can generate is explicitly handled by the firmware. Unexpected faults and interrupts result in a processor reset. As with the power fault, if the engine is running, IntelliJect will restart quickly with key data needed to keep operating the spark and injector. In the event of a software fault IntelliJect will log and report the cause of the fault, the location in code where it occurred, and key processor register values.

In addition, a hardware watchdog is used to make sure the firmware does not get stuck in an infinite loop. The watchdog will reset the processor in the event the firmware does not service the watchdog in a timely fashion.

During bootup, if the engine is not running, the boot process computes a cyclic redundancy check (CRC) of the firmware and compares it against the expected value. If the CRC is incorrect the bootloader will not allow the firmware to start running, preventing the system from operating with corrupted firmware.

6. Configuration fault

Correct operation of the engine is dependent on the correct configuration data; which are stored in two locations in IntelliJect: a factory location, and a user location. At bootup configuration data are set to defaults, then read from the factory location, and then read from the user location. Each piece of configuration data is stored in packet format, including the packet CRC, preventing the use of corrupted configuration data.

An overall configuration CRC is included which can be used to determine if any of the configuration data have been changed since they were last validated. Mismatch of the configuration CRC sets an error bit.
X. ERRORS

As part of telemetry Intelliject reports regular errors and sticky errors. Regular errors indicate problems happening now. Sticky errors indicate problems that happened previously. Some errors only become sticky under certain situations: for example, low fuel pressure will only be a sticky error if Intelliject is enabled (since disabling it disables the fuel pump control).

A. Error windows

There are two different error windows, one for the regular and one for the sticky errors. The Clear button on the sticky errors window instructs Intelliject to clear the sticky errors flags. Once cleared the errors will reassert if the error condition is still present. The Clear button will only be enabled if there are errors that can be cleared.

![Figure 48: Intelliject sticky and regular error windows with and without errors.](image)

Each light in the errors window can represent multiple errors for that subsystem. If you hover your mouse over the window a tooltip will appear giving a summary of all the errors. This tooltip (for the regular errors) is also available in the main toolbar by hovering over the Error light. If the errors window is not visible you can click on the Error light in the main toolbar to make it visible.

![Figure 49: Enable toolbar with an error visible.](image)

Table 1 List of possible error conditions

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage OK</td>
<td>Low voltage detected</td>
</tr>
<tr>
<td>Current OK</td>
<td>Low current detected</td>
</tr>
<tr>
<td>6V Fault OK</td>
<td>6V fault detected</td>
</tr>
<tr>
<td>Injector1 OK</td>
<td>Injector 1 fault detected</td>
</tr>
<tr>
<td>Injector2 OK</td>
<td>Injector 2 fault detected</td>
</tr>
<tr>
<td>Crank1 OK</td>
<td>Crank 1 fault detected</td>
</tr>
<tr>
<td>Crank2 OK</td>
<td>Crank 2 fault detected</td>
</tr>
<tr>
<td>CAN Throttle OK</td>
<td>CAN throttle fault detected</td>
</tr>
<tr>
<td>Speed OK</td>
<td>Speed fault detected</td>
</tr>
<tr>
<td>MAT OK</td>
<td>MAT fault detected</td>
</tr>
<tr>
<td>CHT1 OK</td>
<td>CHT1 fault detected</td>
</tr>
<tr>
<td>dBaro OK</td>
<td>dBaro fault detected</td>
</tr>
<tr>
<td>OAT OK</td>
<td>OAT fault detected</td>
</tr>
<tr>
<td>Test Mode OK</td>
<td>Test mode fault detected</td>
</tr>
<tr>
<td>Maintenance OK</td>
<td>Maintenance fault detected</td>
</tr>
<tr>
<td>Label</td>
<td>Possible error conditions</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Voltage</td>
<td>Input voltage too low or too high.</td>
</tr>
<tr>
<td>12V Volts</td>
<td>12V Voltage too low or too high.</td>
</tr>
<tr>
<td>Current</td>
<td>Input current too low or too high.</td>
</tr>
<tr>
<td>12V Current</td>
<td>12V Current too low or too high.</td>
</tr>
<tr>
<td>6V Fault</td>
<td>6V Fault indicated from power supply.</td>
</tr>
<tr>
<td>Ign Coil</td>
<td>Inductive ignition driver has a fault (open or short).</td>
</tr>
<tr>
<td>Injector1</td>
<td>Injector driver 1 has a fault (open or short).</td>
</tr>
<tr>
<td>Injector2</td>
<td>Injector driver 2 has a fault (open or short).</td>
</tr>
<tr>
<td>Crank1</td>
<td>Initial sync: Crank wheel synchronization error during start Missed edge: two edges in a row of the same type (both high or both low) Overcapture: software did not service the hardware fast enough Too fast: Crank sense active edge was earlier than expected Sync lost: Crank wheel synchronization was lost after the engine was running Too late: Crank sense active edge was later than expected Reversed: Crank is rotating in a direction that is not allowed</td>
</tr>
<tr>
<td>Crank2</td>
<td>Crank sensor 2: Same errors as sensor 1</td>
</tr>
<tr>
<td>CAN throttle</td>
<td>CAN throttle servo is not connected, or servo has a fault.</td>
</tr>
<tr>
<td>CAN cooling</td>
<td>CAN cowl flap servo is not connected, or servo has a fault.</td>
</tr>
<tr>
<td>Speed</td>
<td>Engine speed has exceeded the soft or hard overspeed limit.</td>
</tr>
<tr>
<td>Settings</td>
<td>Parameter settings do not hash to the expected value; a setting may have changed.</td>
</tr>
<tr>
<td>MAT</td>
<td>Manifold air temperature signal is under range or over range.</td>
</tr>
<tr>
<td>MAP</td>
<td>Manifold pressure signal is under range or over range.</td>
</tr>
<tr>
<td>CHT1</td>
<td>Cylinder head temperature 1 signal is under or over range; or, too cold or too hot.</td>
</tr>
<tr>
<td>CHT2</td>
<td>Cylinder head temperature 2 signal is under or over range; or, too cold or too hot.</td>
</tr>
<tr>
<td>dBaro</td>
<td>Digital barometric pressure sensor is not connected; or, signal is under or over range.</td>
</tr>
<tr>
<td>aBaro</td>
<td>Analog barometric pressure sensor signal is under range or over range.</td>
</tr>
<tr>
<td>OAT</td>
<td>Outside air temperature sensor is not connected; or, signal is under range or over range.</td>
</tr>
<tr>
<td>Spare temp</td>
<td>Spare temperature sensor signal is under or over range.</td>
</tr>
<tr>
<td>TPS</td>
<td>Analog throttle position sensor signal is under range or over range, or the throttle position sense is unavailable or inconsistent with other measurements</td>
</tr>
<tr>
<td>Fuel P</td>
<td>Fuel pressure sensor signal is under range or over range.</td>
</tr>
<tr>
<td>Test Mode</td>
<td>Test mode is active.</td>
</tr>
<tr>
<td>CPU</td>
<td>CPU temperature is too cold or too hot. SD card had an error</td>
</tr>
<tr>
<td>Maintenance</td>
<td>If any of the items in the maintenance schedule are due for service. See section VIII.B.</td>
</tr>
<tr>
<td>Starting</td>
<td>If the throttle is higher than Max Start and the engine is not running. See section V.C.3</td>
</tr>
</tbody>
</table>

B. Analog sensor errors

The sensors can have under-range or over-range errors. These errors occur when the sensor is disconnected, or shorted to ground or power. The thresholds that define over- and under-range are part of the sensor calibrations, see section V.I for details. Some sensors have other limits that define errors, for example the CHT sensors have hot and cold errors defined by the engine configuration data; and fuel pressure has a low and high pressure defined by the pump configuration data.
XI. SYSTEM

The system window gives information about the IntelliJect hardware, the firmware, and the CPU performance including reset sources.

A. Firmware

_Firmware_ gives the IntelliJect firmware version. This follows the same format as the version information for the display software. It has a major and minor version number (1.7 in Figure 50), a “Testing” or “Release” indicator, date of release, and a build number. The number in parenthesis is a unique label that tracks the specific source code used to build the firmware.

The display software and firmware are built from the same source tree; and will typically have matching version information. If there is a mismatch between versions software will issue a warning asking if you would like to update the firmware (if the firmware is older).

The _Firmware_ group may also give a description string for the firmware, which may include information about special features of the build (in Figure 50 the description is “1.7 Development”). Lastly, this group displays the cyclic redundancy check (CRC) of the code. The CRC is computed during the firmware build and provides a signature which is unique to the exact size and byte pattern of the firmware.

The IntelliJect bootloader will check the CRC of the firmware when it starts and will not launch the EFI code unless the CRC matches the expected value, to prevent IntelliJect from running corrupted code. The CRC check is bypassed if quick restart is active.

B. Hardware

The Hardware group gives information about the specific IntelliJect hardware. There are three pieces of data configured during manufacturing of the hardware:

- **SN**: gives the serial number of the board.
- **Rev**: gives a revision number of the board. Revision 0 is the prototype hardware used during development.
- **Date**: gives the date of manufacture of the board.
C. Reset and Processing

The *Reset* group gives information about the CPU reset of the IntelliJect hardware. The normal source of reset will be “Power on”, but there can be others: including firmware exceptions, resets for firmware updates, etc.

The first number in parenthesis is a count of the number of times IntelliJect has reset, since it was powered on. The second number is the total number of times IntelliJect has booted up. Figure 51 gives an example of an exception reset, in this case caused by floating point exception. This exception occurred while an engine was running, and therefore Quick Restart was active.

Anytime a non-power on reset occurs IntelliJect will send information in its reset report to describe the state of the processor when the reset occurred. The display software populates this in a message to user, see Figure 52. Unexpected processor resets are not normal, and the IntelliJect firmware is written to be very robust; however, if it does occur this is the information that Power4Flight will need to debug the problem.

The *Processing* group gives information about the real-time performance of the CPU. CPU utilization is not expected to be a problem, but since it is not possible to test all possible combinations of settings this information can be useful in determining if a configuration is overloading the CPU.

- **CPU/Int:** gives the percentage utilization of the CPU for regular code / and for code executing in interrupts.
- **Watch/Int:** gives the maximum time interval between watchdog servicing / and the maximum time spent in any one interrupt.
- **Stack:** gives the percentage utilization of the stack memory.

IntelliJect uses an on-chip watchdog to reset the system if the software becomes stuck in a loop. The watchdog will reset the system after approximately 1 second of no service. Typical watchdog service intervals will be less than 1000 micro-seconds.
XII. SD CARD

The secure digital (SD) card is used for onboard recording of configuration and telemetry data. You should be able to use any modern micro SD card. The system window gives information about the status of the card. In Figure 50 the SD card is healthy (the green indicator), has encountered no errors since the system booted up, and has recorded 18.5 KB of data, out of a total of 5.287 MB of data on a card that can hold 29.711 GB.

A. Card Setup

To use a micro SD card in IntelliJect there are two card setup steps that must be performed:

- Using a PC (typically a windows PC) the card should be formatted using the FAT32 filesystem.
  You can use quick format for this operation, see Figure 53.

- Insert the card in IntelliJect and use the command Setup SD... from the system window. This command will cause IntelliJect to overwrite the root directory file system and file allocation tables. IntelliJect will instantiate a series of files that fill the entire card. During data recording IntelliJect will write into these files.

---

8 There are better file systems, however FAT32 is not encumbered by licensing fees and is readable by multiple PC operating systems including Windows, macOS, and Linux.

9 The biggest problem with FAT32 is the file size limitation ($2^{32} - 1$ bytes). IntelliJect works around this by using multiple files whose data can be concatenated.
Figure 53 shows the sequence of steps needed to use an SD card. On the left is the Windows 10 drive format window; which you can access by right-clicking on the SD card drive letter in Windows Explorer. Most cards will come formatted with some file system out of the box. Smaller cards may already be formatted using FAT32, while larger cards are likely to be formatted with exFAT, which cannot be used with IntelliJect and requires reformatting to FAT32.

Before the card is inserted into IntelliJect the SD Card portion of the system window looks like the upper right picture of Figure 53, indicating no card is detected. When a freshly formatted card is inserted the window looks like the second picture on the right of Figure 53. In this case IntelliJect has detected the card, and that it is formatted in FAT32, but the root directory files have not yet been created.

Once you command Setup SD... you must confirm the command, since this will overwrite the root directory files. The setup process will produce a file system that looks like Figure 54.

The volume ID of the card will be "INTELLIJECT"; and the root directory will be filled with a series of 2GByte read-only files named “SDDATAxy.IJD”. These files are physically laid out on the card in contiguous blocks of memory, so IntelliJect can record data on the card without referring to the file allocation table. From the perspective of a PC operating system the card is full. However, IntelliJect has not actually written any data yet, the file contents are simply whatever was in the card beforehand.

It would be simpler to instantiate a single large file; however, FAT32 limits the file size, hence multiple files are used. Since the files are contiguous on the card the software that reads the files can simply concatenate the data to get the effect of a single large file. IntelliJect does not know the date or absolute time; files are dated using the firmware release date, and file timestamps are set to midnight.

B. Data Recording

IntelliJect records data on the card using communication packets, with special headers to indicate they are SD recording packets. Data are recorded in a continuous fashion: once the entire card is populated with data the recording rolls over, overwriting the oldest data. A typical fast recording rate might use 100K bits per second, which would take 15 days to fill a 16GByte card and begin overwriting old data. Data recording is user configurable, using settings from the communications configuration (section V.J.3) to control the recording rate and when a record is started.
1. What gets written

- Each time a record is started IntelliJect writes all the configuration, maintenance, and logbook data. This provides critical data which will be useful when analyzing engine performance data. IntelliJect will not write any configuration data which is hidden in the locking configuration, see section V.K.1.

- While recording is active any command (via CAN, UART, or USB) sent to IntelliJect, and any response, is written on the card.

- While recording is active the telemetry outputs are recorded. The recording rate is specified in the communications, see section V.J.3, but can be different from normal communications rate. Telemetry makes up the bulk of information that will be written to the card.

![SD Card Status](image)

Figure 55: SD Card status when the card is configured and recording

When IntelliJect is recording the SD card status data will indicate the number of card errors, the number of records on the card, the amount of data that has been recorded in the current record, the total amount of data recorded, and the total amount of data the card can hold. You can insert or remove an SD card while IntelliJect is powered up. Such an event will be seen by IntelliJect as a card error; and will be gracefully handled if the card has been correctly setup.

2. Recording format

SD cards record data in blocks of 512 bytes. The first block (first 512 bytes of the first file) and the last block (last 512 bytes of the last file) always record a special journal packet. The journal documents the location and size of the records on the card; and is recorded twice for redundancy.

Each recording is identified by an incrementing record number. Within a record of data each block has an incrementing sequence number. The record number and sequence are written to the start of each block in a small packet, and together identify where that block’s data belongs. The remainder of the block simply receives bytes of data that represent packets being recorded to the card.

Together the journal and the block sequencers provide enough information to take the data from the SD card and recreate one or more files of data that can be replayed in the display software, see section XXVI.B for details on file replay. The journal contains enough space to document the most recent 81 records on the card. If there are more records than this the old records can be retrieved, but only by reading the entire card contents.
C. Data Retrieval

There are two ways to retrieve data from the SD card. The fastest method is to physically remove the card from IntelliJect and plug it into a PC. Alternatively, you can retrieve data through one of the communications interfaces (USB, UART, CAN).

1. Moving the card to a PC

The card can be removed from IntelliJect and inserted into a PC. Then you use the File menu command Parse SD Card.... This will prompt you to select a SD card file. It does not matter which file you choose; the display software will access all the SD card files (files ending in “.IJD”) in that directory.

You must keep the SD card files together, and not mix files from one SD card with files from another. You can copy files from an SD card to your computer, if you copy all of them to an otherwise empty directory. Don’t change or write to a file on the card after IntelliJect has set it up.

The display software will read the journal information from the files and ask which record you would like to retrieve. You can choose to retrieve a single record, or all the records on the card. The selection window orders the records from most recent to oldest (in the journal) and displays the size of the records.

The journal can only hold 81 records of information, even though the card could hold many more. If you need to access a record which is older than all the journal entries, you must parse the oldest record on the card (or all the records on the card).

After you have selected the record the display software will read through the files, extracting packets of data that belong to that record, creating both a replay file (.efitel) and a text file (.csv). These files have the same format as the log files that are normally written by the display software, see section XXVI for more details.
2. Using communications

To transfer a record of SD data over a communications channel use the **Transfer...** button on the status page. This will present a dialog box asking you to select a record to transfer. You can only transfer one record at a time using this method.

![Figure 59: Selecting an SD record for transfer](image)

While the transfer is running the system will not perform any SD recording, and it will temporarily reduce the communications rate to 1Hz, in order to preserve bandwidth for the SD transfer. The transferred data is used to create both a replay file (.efitel) and a text file (.csv). These files have the same format as the log files that are normally written by the display software, see section XXVI for more details.

![Figure 60: SD Transfer finished](image)
XIII. **DISPLAY LAYOUT**

The display has a user adjustable layout. Windows and toolbars can be moved, resized, tabbed, or hidden to suit the user and available display space. Windows and toolbars can also be undocked if desired. This behavior is mouse driven and reasonably easy to discover by simply experimenting.

Once you have settled on a display configuration you can save it to a file using the menu Windows->save display layout, reloading the file configuration later using Windows->open displays layout. However, you don’t need to save and load display layouts, the software will automatically remember it for you (on a per-user basis).

If your display layout becomes too confusing use the Windows->default display layout feature to return to the default layout (given below in Figure 61). You can also prevent layout changes by using Windows->lock display layout.

![Figure 61: Default display layout.](image)
XIV. **GAUGES**

The gauges provide a graphical look at the performance of the engine. The graphical displays are colored to indicate status. For example, the CHT will be blue if it is too cold, red if too hot, and otherwise green. The text at the bottom of the display gives the current operating mode of IntelliJect. On the left of Figure 62 IntelliJect is holding a constant throttle commanded by the user; on the right, it is controlling an RPM that was commanded by the user.

![Figure 62: Gauges.](image)

The default data layout is shown on the right of Figure 62, but it can be changed by right-clicking on the label of any element and selecting or removing a variable. Available variables are the same as for strip charting (section XV), however the displayed names are compressed to save space.

The selection drop down also gives you the option to remove a signal, or to insert a signal below the label you clicked. You can have up to 20 text items in the gauges display (10 on the left column, 10 on the right). The text layout is saved automatically with the rest of the display layout, and can be saved and loaded explicitly, see section XIII.

![Figure 63: Variables.](image)
Charts are used to provide a rolling plot of IntelliJect telemetry data against time. Each chart can display two different axes of data (left and right), and each axis can have multiple data variables. In Figure 64 Throttle, CHT, and Inj 1 duty are on the left axis, and RPM is on the right axis. The horizontal axis gives time in hours:minutes:seconds since IntelliJect turned on.

Use the Select Signals... button on the chart to bring up the chart configuration dialog (see Figure 65). Use the Add Signal... drop down to add a signal to the chart. The signals available for plotting are the same as those which are written to the human readable log file (section XXVI). The list of
signals, their colors, axis assignments, and axis scales are automatically saved by the software. Chart configurations are also preserved in layout files.

The chart signals window shows the signals that are already selected for the chart in a table. To change line width, colors, or remove signals select one or more rows in the table and use the Width..., Color..., or Remove buttons respectively.

A. Adjusting time scale

The scroll bar at the top of each chart window can be used to move backward and forward in time. Time scrolling can be done while new data are accumulating in the chart. Scrolling one chart will cause all charts to scroll to match the time axis if Sync is checked. Scrolling to the far right will restore the charts to real time operation. You can adjust the time scale in multiple ways:

- The scroll bar can be used to adjust the position of the time axis.
- Click and drag (right or left) on the time axis works the same as the moving the scroll bar.
- Double clicking on the x axis of the chart will restore the chart to real time operation.
- The magnifying glass buttons between the scroll bar and the Select Signals... button will zoom the time axis in or out.
- Using the mouse wheel (or two-finger scroll on a trackpad) over the x-axis will zoom the time axis in or out.
- The Time value at the bottom of the chart signals dialog will adjust the time scale.

B. Adjusting left and right scales

The left and right y axis scales can be set manually or automatically. The Auto buttons above the left and right vertical axes toggles the axis to be automatic or manually scaled. In Figure 64 the left axis is manual, and the right axis is automatic. When an axis is automatically scaled the buttons next to the axis are hidden. You can adjust the vertical axis scales in multiple ways:

- The Auto button is used to toggle between manual and automatic scales. Note that the automatic scale uses all the chart data, not just the data that is currently visible.
- Double clicking on the vertical axis will toggle manual or automatic scaling.
- The magnifying glass buttons next to the vertical axis will zoom the axis in or out.
- The arrow buttons will shift the data of the axis in the direction of the arrow.
- The button between the arrows will shift the scale to center the chart data.
- Click and drag (up or down) on the vertical axis shifts the data up or down, if the axis is not automatically scaled.
- Using the mouse wheel (or two-finger scroll on a trackpad) over the vertical axis will zoom the axis in or out, if the axis is not automatically scaled.
- The axis limits can also be set from the chart signals dialog.

C. Buffer size

The chart signals are stored in computer memory even if the signal is not currently charted. This makes it possible to add a signal and see the history of that signal, including data from before the
signal was added to the chart. The size of the memory buffer is configurable using the **Buffer Size**
dropdown. Small buffers reduce the memory and CPU resources used by the computer; however
large buffers allow you to scroll further back in time. If you run the software for a long time with
large buffers the computer may become slow, particularly if many points are displayed on a chart.

The default buffer size is 100 thousand points, which will hold data for 83 minutes if the signal in
question is updated at 20 Hz. Once the buffer is full the oldest data is discarded from memory.
Note that all the data are always saved in the log files and can be replayed to regenerate a chart.
The largest available buffer size is 10 million points, which is likely to be troublesome on all but
the most powerful computers.
XVI. COMMAND TOOLBARS

A. Enable toolbar

IntelliJect has five enable signals which are visible in the toolbar. IO Enable is not controllable: it indicates the status of the input line which can be used to disable IntelliJect, see section XXVI. User Enable is a button which allows the user to enable/disable IntelliJect. Both IO Enable and User Enable must be green for IntelliJect to be enabled. If disabled the fuel pump output will be off, the injectors will not open, and the sparks will not fire.

Ign 1 Enable, Ign 2 Enable, and Ign 3 Enable control the first, second, and third ignition outputs respectively. These can be independently disabled to verify correct operation of each ignition system. The third ignition output is shared with the first cooling flap PWM output, and must be enabled in the engine configuration (section V.E.4) for the button to appear in the toolbar.

![Image of toolbar with enable signals](image)

Figure 66: Enable toolbar with all enabled and with all disabled, spark 3 enabled.

The indicator to the right of Ign 3 Enable is a summary of the IntelliJect error status, see section VI for more information. At the right of the toolbar is the elapsed time since IntelliJect turned on.

B. Throttle and RPM toolbars

These toolbars are used to send throttle and RPM commands. They are only enabled if IntelliJect is configured to drive the throttle, see section V.C. Buttons on the toolbar provide quick access to common throttle and RPM commands. You can also enter the throttle or RPM command directly in the text boxes. When not being used to send commands the text boxes display the current throttle and RPM telemetry.

The toolbars are populated with throttle and RPM command buttons taken from the fuel table indices. If a throttle curve is enabled the throttle command bar will include the Outs checkbox. Set this check to apply the throttle curve in reverse before sending the command. This will cause the throttle output (rather than the input) to follow the button command.

The toolbars are normally docked vertically at the side of the software, but they can be docked horizontally at the top or bottom.

![Image of throttle and RPM toolbars](image)

Figure 67: Throttle and RPM toolbars.
XVII. **PROFILE RUNNER**

The profile runner is a tool for exercising an engine according to a predefined profile of throttle and/or RPM commands.

1. **Definition of the Profile**

The left side of Figure 68 gives a table that defines the profile. Each row of the table is a step in the profile. The table columns are:

- **Start[s]**: is the start time of this step profile, relative to the start of the entire profile. The start time cannot be edited; it is computed based upon the individual step times.
- **Cmd**: is the command value for the step. Values 100 or less are interpreted as throttle commands, otherwise the values are interpreted as RPM commands.
- **Times[s]**: provides the amount of time in seconds used by the step.
- **Direct**: is a flag to control how the command is given. If set the command goes immediately to the value when the step begins. If clear the command moves in a straight line from the previous step’s value to the new step’s value.
- **Expected**: gives the RPM the engine is expected to turn at this step. The expected RPM is only useful for throttle command steps, not RPM command steps. Expected RPM is used as part of the scoring system for acceptance test reporting, see section XXIII.B.1.
- **Enable**: gives the user enable status at this step. Clearing this checkbox will stop the engine from running at this step.
- **Ign1**: gives the ignition 1 enable status at this step.
- **Ign2**: gives the ignition 2 enable status at this step.
- **Ign3**: gives the ignition 3 enable status at this step.

![Figure 68: Profile Runner table and chart tabs, showing a running profile with mixed throttle and RPM.](image)

The shape of the profile is visible in the thin lines on the chart (black for throttle, blue for RPM). To add or remove a step from the profile right-click on the table. Profiles can be loaded from, or saved to, a file using the **Load**... and **Save**... buttons. The profile file name is displayed at the upper left of the window. Profile files have an extension “.efipro”; and simple human-readable space
(or comma) delimited content. If no profile was previously loaded the display software loads the *Standard Checkout* profile, which can be found in ~/Power4Flight/IntelliJect Display/profiles/. Load this file in a text editor to see the profile file format. The profile can also be edited in the table by double-clicking the cell you want to change. Right-click the table to insert or delete rows.

2. Creating a profile

You can create a new profile using the **Create...** button, which will show the profile builder window, see Figure 69. The profile builder window provides a simple way to build a profile like the standard checkout profile. This is often the easiest way to build a new checkout profile.

![Profile Builder Window](image)

Figure 69: Profile builder window.

- Check the **Throttle** box to include the portion of the profile that exercises the throttle steps.
- Check the **RPM** box to include the portion of the profile that exercises the RPM steps.
- If **Match fuel table** is checked the steps of the profile match the indices of the fuel table that are within the *Min* and *Max* values.
- If **Split cells** is checked the match table feature uses values halfway between the table indices.
- If **Match tables** is not checked the *Min*, *Max*, and *Steps* values define the shape of the throttle and rpm sections of the profile.
- **Time** specifies the seconds spent at each step.

3. Running a profile

To start a profile, press the **Start / Stop** button. As the profile runs the current position in the profile table is marked by a red line; and the throttle input, output, and RPM, are plotted on the chart using thick lines (magenta for throttle input, black for throttle output). If you stop a profile it will resume its operation at the same point when restarted. To change this behavior the **GoTo...** button can be used to set the current profile time; or right-click on the chart or the table to go to a specific profile time. Use the **Repeat** option to have the profile automatically start over when it reaches the end.

The profile runner works by sending throttle or RPM commands and enable commands. Therefore, it is only useful for EFI systems that have control over the throttle, see section V.C for more information. If you use the throttle or RPM command toolbars to send a command the display software will automatically stop a running profile. The same is true if the alarm system performs an engine shutdown, or if you use the main toolbar to send an enable command. Throttle commands affect the input throttle, the output throttle may be different if a throttle curve is active. If you select the **Outs** option from the throttle toolbar (section XVI.B) the profile runner will apply the throttle curve in reverse so the commands yield output throttles.
XVIII. ALARMS

The alarm window provides a configurable system to alert the user of problems. This is not a function in IntelliJect, it is purely a safety and alert system for the user of the display software.

Use the Enable Alarms checkbox to globally enable or disable all alarms. Use the Alarms Silent checkbox to enable or disable the audible alarm sound. The audible alarm sound will be inhibited if Silent if Stopped is checked and the engine is not running. The Shutdown Enable checkbox enables automatic shutdown from critical alarms (by issuing a throttle close and user disable command). Shutdowns do not happen immediately, there is a time window to fix the problem or acknowledge the alarm. Only the CHT critical alarm can trigger a shutdown.

![Alarm display with and without alarms active.]

Individual alarms are enabled or disabled with the checkbox next to the alarm name. If an alarm activates its button will turn red with white text. Acknowledge the alarm by clicking on the button, which turns off the alarm (and changes the button color to white with red text). An alarm will remain in the acknowledged state until the alarm condition clears. The Ack All button acknowledges all active alarms.

CHT and fuel pressure alarms have configurable thresholds. Some alarms have special arming conditions, for example CHT cold and Fuel pressure low alarms will not become active until the temperature and fuel pressure have gone above the threshold for the first time. This prevents alarming when the system is first turned on. Similarly, EFI communication lost and Fan communication lost alarms will not trigger until communications are first established.

A summary of the alarm status is in the alarm toolbar. The toolbar button acknowledges all alarms, and the Silence checkbox enables the audible alarms. You can click on the Alarm label in the toolbar to make the alarm window visible.

![Alarm toolbar with no alarms, alarms active, and alarms acknowledged.]

Figure 70: Alarm display with and without alarms active.

Figure 71: Alarm toolbar with no alarms, alarms active, and alarms acknowledged.
The packet log window gives a display of all packets sent and received by the display software. This is a useful diagnostic tool for understanding or debugging communication to or from IntelliJect.

Each line of the display gives the system time (in milliseconds) of the packet, the direction of travel (From or To IntelliJect), the packet type enumeration, the number of data bytes in the packet, and description of the packet.

In Figure 72 you can see a request for all configuration data, the responses from IntelliJect, followed by a throttle command and an RPM command.

You can scroll the list vertically in time if the number of packets exceeds the size of the display. Use the Clear button to clear the packet log display.

Since IntelliJect is constantly sending telemetry packets the log is quickly overwhelmed by the telemetry. Use Hide Telemetry to mask the telemetry packets from the log display.
XX. **Table Visualization**

The table visualization window provides multiple ways to look at a table. It is used as a complement to the table configuration window (section V.B), which selects the displayed data.

The upper plots in Figure 74 are the surface plots. On the left is a “heat map” which maps the table values to a color. On the right is a 3D surface map, which uses the same coloring as the heat map, but allows you to visualize the table values as a surface. Click and drag with the mouse to show different perspectives on the 3D surface. The surface plots can be hidden by clearing the **Show Surface** checkbox.

The lower plots in Figure 74 gives the table values on as a function of either the column indices (on the left) or the row indices (on the right). The line plots can be hidden by clearing the **Show Plots** checkbox.
If the **Col origin**, **Row origin**, and **Table origin** checkboxes are checked the plots are forced to include the zero point for the respective dimension. Figure 75 gives an example of showing a subset of a table while forcing the origins to be included.

![Visualization of a table subset](image)

**Figure 75:** Visualization of a table subset.
XXI. OSCILLOSCOPE

The oscilloscope is an advanced feature used to capture analog and digital signals with high time resolution. This provides visibility into key timing information and intra-cycle variation of sensor data. Some of the signals (for example the analog sensors) are synchronously sampled at 1kHz; and some of the scope signals are sampled asynchronously when the digital state changes.

The scope can be run triggered or untriggered. The triggered mode of operation is the most common, and typically the crank sense signal is used to trigger the scope. This mode of operation is useful for visualizing the spark and injector timing. The scope can output a single trace of information or it can continuously output traces until you tell it to stop.

Figure 76: Oscilloscope window showing a scope capture.

A. Scope setup

The scope configuration is volatile, so a reset of IntelliJect will stop any scope acquisition or output. Configure the scope to the desired setup before pushing the Start button.

- Press the Start button to begin the process of scope data acquisition and transmission. You can press this button while the acquisition is in process to stop the scope.
- Select the Once checkbox to generate a single scope trace. If the Once checkbox is clear the scope will immediately begin a new acquisition when the previous one is output.
- The time next to the Once checkbox gives the duration of the scope trace. Larger durations will require more data buffering and more time to transmit the result. IntelliJect may reduce the duration if you have too many channels selected to fit in the buffer.
- The Trigger:Chan drop down selects which channel is used to trigger the scope.
- Trigger:Level sets the signal level at which the scope triggers. Digital channels only have two levels: 0 and 100%.
- Select the Trigger:Falling check box to trigger the scope when the trigger signal crosses the trigger level from high to low. Otherwise the trigger signal must cross the trigger level from low to high.
Once the oscilloscope has begun to output data the progress bar below the setup will show the progress of transmitting the data to the display. The time stamp below the progress bar is the start time for the scope acquisition.

B. Scope signals

The Add Signals... button displays the scope signals window, see Figure 77. This window is like the strip chart signal selection (see section XV) except for the channels, see Table 2.

The scope signals window is used to choose the channels to acquire and plot on the scope chart. Channels should be selected before turning the scope on. Since digital signals take very little space (two bytes for all the digital signals) the scope always acquires all the digital channels no matter what is selected. However, the scope will only acquire analog channels if they are selected. Therefore, you can add digital signals after a scope trace is received, but you cannot add analog signals to a trace that has already been received.

![Scope signals window and list of channels.](image)

**Table 2 List of scope signals.**

<table>
<thead>
<tr>
<th>Name</th>
<th>A/D</th>
<th>Sync</th>
<th>Signal description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Voltage</td>
<td>A</td>
<td>Sync</td>
<td>Main input voltage in Volts</td>
</tr>
<tr>
<td>MAT</td>
<td>A</td>
<td>Sync</td>
<td>Manifold air temperature in °C</td>
</tr>
<tr>
<td>CHT1</td>
<td>A</td>
<td>Sync</td>
<td>Cylinder head 1 temperature in °C</td>
</tr>
<tr>
<td>CHT2</td>
<td>A</td>
<td>Sync</td>
<td>Cylinder head 2 temperature in °C</td>
</tr>
<tr>
<td>MAP</td>
<td>A</td>
<td>Sync</td>
<td>Manifold air pressure in kPa</td>
</tr>
<tr>
<td>Fuel pressure</td>
<td>A</td>
<td>Sync</td>
<td>Fuel pressure in kPa</td>
</tr>
<tr>
<td>Analog TPS</td>
<td>A</td>
<td>Sync</td>
<td>Analog throttle position sensor in %</td>
</tr>
<tr>
<td>CPU Temp</td>
<td>A</td>
<td>Sync</td>
<td>Manifold air temperature in °C</td>
</tr>
<tr>
<td>Spare Temp</td>
<td>A</td>
<td>Sync</td>
<td>Spare temperature in °C</td>
</tr>
<tr>
<td>Current</td>
<td>A</td>
<td>Sync</td>
<td>Main input current in Amps</td>
</tr>
<tr>
<td>12V Volts</td>
<td>A</td>
<td>Sync</td>
<td>12V voltage in Volts</td>
</tr>
<tr>
<td>12V Current</td>
<td>A</td>
<td>Sync</td>
<td>12V current in Amps</td>
</tr>
<tr>
<td>Barometer</td>
<td>A</td>
<td>Sync</td>
<td>Analog barometric pressure in kPa</td>
</tr>
<tr>
<td>IO Enable</td>
<td>D</td>
<td>Sync</td>
<td>Enable input state</td>
</tr>
</tbody>
</table>
1. **Synchronous signals**

When the scope is enabled IntelliJect makes a scope sample of all the signals at 1kHz. These samples are called synchronous because the signals are sampled at effectively the same time. Analog signals are only available synchronously. Some digital signals are also only available synchronously, see the “Sync” flag in Table 2.

2. **Asynchronous signals**

Some of the digital signals are also sampled asynchronously; which means that the scope sample is taken when the signal changes state. This applies to signals flagged “Async” in Table 2. Asynchronous sampling is important since many of the digital signals change very quickly, and the synchronous sampling would not capture the interesting details of these signals. The last channel “Asynchronous” is not real a signal, it is instead a digital sample bit indicating if the other signals in that scope sample were acquired asynchronously or synchronously.

C. **Scope chart**

The scope chart is like the strip chart (see section XV); however, the x-axis time is with respect to the time of the trigger event, or the start of the scope acquisition. For triggered acquisitions IntelliJect acquires half the time interval before the trigger, and half the time interval after the trigger.

The scope chart does not automatically scroll the time axis, instead the chart data are discarded and replaced each time a scope acquisition is received. As with normal strip charts you can save a log of the traces on the scope chart using the **Save**... button.
XXII. **TEST MODE**

Test mode is used for hardware debugging and test. It is only available when the EFI is unlocked. Test mode can simulate a crank signal, run injectors, ignitions, fuel pump, calibrate servos, introduce errors and test the user storage.

![Figure 78: Test mode window with test not running and running.](image)

- **Start Test**: is used to activate timed tests (All tests except for Skip Sense, Add Sense and Missfire tests). If a test is already running this button will say “Refresh Test”, and it will reset the test setup, resetting the test countdown timer.
- **Stop Test**: is used to stop test mode if it is active.
- **Time**: gives the duration of the test in seconds. If the test is running the elapsed test time is displayed in the progress bar to the right of the duration.

### A. Output tests

The output tests are used to test injector, spark, and fuel pump outputs. Output tests are automatically canceled if the engine starts running, or if Time elapses.

- **Speed**: gives the frequency of test events per minute.
- **Duty**: gives the duty cycle of the test output in percent.

#### 1. Simulated crank sense

If **Crank Normal** or **Crank Reverse** is checked the system simulates crank sense events in normal or reverse direction. The crank sense events occur at the **Speed** frequency. **Duty** specifies the amount of time the simulated crank signal is active. Normal EFI computations (fuel, advance, etc.) will be performed exactly as if real crank sense events were occurring. In addition, the injector and spark outputs will behave exactly as with real crank signals. The simulated crank sense will be displayed in the oscilloscope, even though the actual crank sense inputs are not changing state.

The crank sense test cannot be performed at the same time as other tests. If you have other tests selected alongside the crank sense test the other tests will be automatically canceled. If a crank wheel is configured the crank sense events will follow the specified crank wheel design. The crank sense test only applies to crank sense 1.
2. Fuel pump test
If Fuel pump is selected the system runs the fuel pump output at the duty cycle specified by Duty. The frequency of the fuel pump output is not affected by the Speed setting (fuel pump frequency is controlled by fuel pump configuration settings). The fuel pump test can be run simultaneously with the injector, spark, and fuel flow tests.

3. Injector test
If Injector1, Injector2, or Injector3 is checked the selected injectors are operated at the frequency and duty cycle given by Speed and Duty. The duty cycle of the injector output is exactly specified by Duty, with no adjustment for the injector trim. Instead the injector trim is accounted for in the computation of the effective injector duty cycle, injector time, and fuel flow rate in the telemetry.

The injector tests can be run simultaneously with the spark tests, the fuel pump test, and the fuel flow test. If the fuel flow test is active the injector duty cycle will be controlled by that test.

4. Spark test
If Spark1, Spark2, or Spark3 is checked the selected ignition outputs are operated at the frequency and duty cycle given by Speed and Duty. The duty cycle specifies the amount of time the ignition output is in the active state. The spark test will ignore the minimum ignition active time. The spark test can be run simultaneously with the injector test, the fuel pump test, and the fuel flow test.

5. Fuel flow test
The fuel flow test is engaged if Fuel flow is checked. This test runs a proportional-integral feedback control law from fuel pressure to injector duty cycle. The control law attempts to keep the fuel pressure at the value set in the Intelliject configuration (see section V.H) by changing the duty cycle of the injectors.

- Fuel flow: Pro Gain: gives the proportional feedback gain, in units of percentage of injector duty cycle per kPa of pressure error.
- Fuel flow: Int Gain: gives the integral feedback gain, in units of percentage of injector duty cycle per kPa-seconds of integrated pressure error.

The fuel flow test requires that an injector be selected. If no injector is selected in the test configuration Injector1 is automatically selected. The fuel pump output is operated at Duty if Fuel pump is checked, or 100% output if Fuel pump is not checked.

B. Crank error test
There are two types of crank error tests: skipping or adding sense events, which are not timed tests, and suppressing crank sense 1 or 2, which is a timed test. In any case the crank error tests require that the engine be running.
• Suppress sense 1 or 2: Select this option and push **Start** to suppress the use of a crank sensor for scheduling spark and injector outputs. The sensor will still be used to compute RPM and crank synchronization. Suppressing the sensor allows you to see what the engine performance will be like if the sensor fails and the system reverts to the other sensor.

• **Skip Sense 1 and 2**: push this button to skip a single crank sense event for sense 1 or sense 2. This test is used to evaluate the system performance when a crank sensor is not reliable. It can be used with once-per-rev or crank wheel sensors.

• **Add Sense 1 and 2**: push this button to add a fictitious crank wheel tooth. This test can only be used with crank wheel sensors.

C. Calibrate servos against hard stop

If **Throttle closed, Throttle open, Cowl 1 closed, Cowl 1 open, Cowl 2 closed, or Cowl 2 open** is checked test mode will drive the respective servo PWM output signal to find a hard stop, calibrating that servo endpoint. The test can only be started if the system is disabled. The test drives the servo, monitoring the input current until it increases above **Current Threshold**, then drives the servo back the other way until the current is reduced to 25 mA above the starting current. The resulting PWM is recorded as the closed or open PWM (depending on which test was run). When calibrating the closed position, the open PWM is also adjusted to keep the PWM span unchanged. The test determines the direction to drive the PWM from the throttle or cowl flap configuration before the test is run.

If **Actual closed stop or Actual open stop** are not 0% or 100% respectively the PWM that detects the hard stop is adjusted to account for the difference between the actual position and 0% or 100%. This makes it possible to, for example, put the throttle hard stop at 105% while still determining the PWM for 100%.

D. Internal tests

These tests are different from the other tests, they are not activated using the **Start Test** button, and they do not have a timeout, speed or duty cycle.

1. User storage

The user storage test is activated with the **User Storage** button. Once the test is activated every byte of user nonvolatile parameter storage is tested. If the test passes the user storage will be unchanged. If the test fails, or if you disconnect power while this test is running, the contents of user storage may be corrupted. Be sure to save your settings to a file (or to factory storage) before running this test.

The test is tracked by percentage of the storage that has been tested, and the percentage completed will be displayed at the top of the test window while the test is running. At the conclusion of the test you will get a dialog box informing you of the results of the test.
E. Misfire tests

The misfire test is used to skip a spark output, effectively creating a misfire. It is not a timed test, and requires the engine be running. You can skip one, two, or three successive spark outputs, for any of the three ignition outputs.

- **Spark1**: selects the first spark output to be skipped with the One, Two, or Three button.
- **Spark2**: selects the second spark output to be skipped with the One, Two, or Three button.
- **Spark3**: selects the third spark output to be skipped with the One, Two, or Three button.
- **One**: causes the selected spark outputs to be skipped once.
- **Two**: causes the selected spark outputs to be skipped twice.
- **Three**: causes the selected spark outputs to be skipped three times.
You can use IntelliJect to produce an acceptance test report that gives key details about the engine and EFI integration. Typically, the report would be generated following the checkout of a new engine, to document the engine performance and configuration.

Figure 79: Acceptance test report window before being filled out.
The acceptance test report is generated when you select the menu item **File->Acceptance Test Report**. At the time the report is generated any visible charts are copied into the report. In addition, important configuration data are copied into the report, see Figure 79. The test report window is forced to an aspect ratio of 8.5 x 11 so it is suitable for printing.

A. User provided data

The report has fields that you must fill out before it can be saved:

- The engine type or name
- The name of the operator who performed the acceptance test
- The name of the IntelliJect configuration file (click on the **Config file:** label to get a dialog for selecting the configuration file)
- The propeller used in the checkout
- The direction of rotation of the propeller (pusher, tractor, both)
- The fuel pump type or name
- The customer order information

The **Save** button will become active after this data have been input, see Figure 80. If the **Save** button is not active you can hover over the button to get a tooltip explaining what data are missing. The button asks you to specify a file which will record the acceptance test report as a png image. If you want to change the configuration you can exit the report window, change configuration, and then generate a new report. Any data input from before will be automatically re-entered in the report. Input data are forgotten when you exit the display software (This is intentional – to prevent accidentally generating a report with stale data).

![Acceptance Test Report](image)

Figure 80: Acceptance test report after putting in required data.
B. Acceptance test scores

If the chart data include changing throttle commands the acceptance system will generate a set of scores for the engine performance. Scores are generated for every throttle command transition. For example, the chart in Figure 79 shows 32 separate throttle transitions (on the left side of the chart, the right side of the chart are RPM commands – not used for scoring).

Each transition, whether increasing or decreasing throttle, is graded according to five separate criteria. Push the Scores: button on the acceptance test report to get the score data for each transition, see Figure 81. You can save the scores, and the information used to compute them, by pushing the Save Scores button.

1. Power

The second half of each constant throttle section is used to compute an average RPM (the first half is not counted as the engine speed may still be settling). The power score is computed using the average RPM and the expected RPM from the profile, see section XVII. If the average RPM matches or exceeds the expected RPM the power score is 100. If the average RPM is 80% or less of the expected RPM the power score is 0. If no profile is loaded, or if the profile does not have expected RPM data, the power score is 100.

2. Stability

Using the second-half data the standard deviation of the RPM is computed. If the standard deviation of the RPM is 0, the stability score is 100. If the standard deviation of the RPM is 10% or more of the average RPM, the stability score is 0.

3. Overshoot

For increasing throttle overshoot is the maximum RPM (of the entire time interval) minus the second-half average. For decreasing throttle overshoot is the second-half average RPM minus the minimum RPM. The overshoot score is 100 if the overshoot is 0; and 0 if the overshoot is 20% or more of the second half average RPM.
4. Responsiveness (lag)

The lag score measures the elapsed time for the RPM to get within 200 of the second-half average RPM. If the lag time is 0.5 seconds or less the score is 100; and if the lag time is 2.5 seconds or more the score is 0.

5. Transition

The transition score works by accumulating the amount of wrong-way RPM changes versus right-way RPM changes. For increasing throttle the RPM is going the wrong way if it decreases, and vice versa. The accumulation of the up-going and down-going RPM changes does not begin until the RPM has moved by 200 from its initial value, and the accumulation stops when the RPM is within 200 of the second-half average RPM. The transition score is 100 if there are no wrong-way RPM changes. The transition score is 0 if there are wrong-way RPM changes that accumulate to 20% of the right-way RPM changes.


The acceptance test report gives a summary of all the scores. The summary is assembled from the worst score in each of the five categories. For the power score, only the wide-open throttle cases are considered for the summary.
XXIV. SIMULATOR

The simulator is a simple tool that can be used to explore the IntelliJect system without a running engine or hardware. The simulator is considered one of the connection methods, and is launched from the IntelliJect connection window, see section III.A.5. If the simulator is running it will be indicated in the status bar. In addition, the simulator window will be enabled for display.

The simulator window provides rudimentary controls to affect the simulation. The IO Enable check box is used to simulate the enable input line. The Crank1 and Crank2 check boxes enable the simulated crank 1 and crank 2 sense events.

The RPM displays shows the engine speed of the simulation, and the Time is the amount of time the simulation has been running.

The simulator is built from much of the same source code as the IntelliJect firmware. It is an extremely useful tool for learning about the system, and for debugging software written to work with the IntelliJect. However, don’t make the mistake of assuming the simulator is a full engine simulation, it is not. For example, the simulator will not respond like a real engine to changes in fuel or spark settings. To start the simulation running you need a throttle command above 10%, which will cause the simulator will crank the engine if it is not already turning.

The simulator emulates the user and factory storage configuration storage with the files efiStorageEmulation.dat and efiFactoryStorageEmulation.dat in the “~/Power4Flight/IntelliJect Display/” directory. If these files are missing the simulator will create the factory storage file with appropriate settings for the simulator.

---

10 Indeed, most of the IntelliJect software was written against the simulator before any hardware existed.
XXV. Firmware Update

A. Firmware distribution

The firmware for IntelliJect is distributed as part of the display software. When the software connects to IntelliJect it will compare the version information to the display software version. If the versions do not match the software will ask if you want to upgrade IntelliJect (if display is newer) or recommend that you update the display software (if IntelliJect is newer).

If you select the Do not ask again for these versions option the display software will remember that you do not want to be bothered again, however that choice applies only to this specific combination of EFI and display version numbers. If either the display or EFI version numbers change in the future you will be asked again.

If you select OK to the firmware upgrade request (or use the menu: About->Firmware Update...) you will get the IntelliJect firmware programming window, see Figure 84. Normally you will see this window only after already connecting to IntelliJect, so the window will indicate that it is already connected. In the Connection group of Figure 84 you can see the programming tool is connected, using port COM25. You can use the Port drop down to change the connection.

The File group shows information about the firmware file. In Figure 84 the firmware file is “:/efifirmware.hex”, which is the file embedded in the display software. This file will be selected...
automatically by the display software, but you can choose a different file using the **Open** button. The **Version**, **Size**, and **CRC** data in the **File** group are read from the contents of the firmware file.

The IntelliJect group shows information about the connected IntelliJect. The **Version**, **Size**, and **CRC** data in this group are sent in response to a request from the programming tool. If these data are identical to the data in the file there is no reason to program.

![Image](image1.png)

**Figure 85**: Programming in progress, connected to Bootloader.

### B. Programming the firmware

Push the **Program** button to start the programming process. The process starts by commanding a reset into bootloader mode. The bootloader sends a periodic telemetry message, and once the programming tool detects the bootloader it will command it to erase the on-chip flash\(^\text{11}\), and then begin sending program data to write the flash. During this time, the programming tool will look like Figure 85. Once programming is complete the programming tool will command the bootloader to reset again, which will trigger the code to begin executing.

![Image](image2.png)

**Figure 86**: IntelliJect in bootloader question.

If programming fails (for example if power is lost) IntelliJect will be left in state in which it cannot run. However, the bootloader will be unaltered, and it will detect the invalid program CRC and remain in bootloader mode, sending bootloader telemetry messages. If the display software gets one of these messages, it will ask if you want to launch the programmer to reprogram the firmware.

The bootloader is a separate program loaded onto the CPU of IntelliJect. Changing the IntelliJect firmware does not change the bootloader; the bootloader can only be changed at the factory. After reset into bootloader mode you can see the version of the bootloader. In Figure 85 the

\(^{11}\) The erase operation does not erase the sectors of flash that store the bootloader, factory storage, or the logbook.
bootloader is version 1; which supports programming firmware over USB or RS-232, but not CAN. CAN firmware programming is supported by bootloaders with version 2 or higher.

C. Bootloader communications over USB

Programming over USB is complicated by the CPU reset process. The USB peripheral on IntelliJect requires that the firmware cooperate in the USB enumeration process of the host operating system (windows, mac, linux). As a result, the reset to bootloader (and bootloader back to IntelliJect) effectively disconnects the USB interface. For this reason, each time the programming tool commands the reset it will shut down the USB port and wait three seconds before attempting to re-open it. This process could be interrupted if the port name in the operating system driver is not what is expected, or if you use a virtual machine to run the operating system. If you have trouble programming firmware over USB, try the UART or CAN interface.

D. Bootloader communications over UART

Because the bootloader is a separate application from IntelliJect it does not have access to the IntelliJect communications configuration (see section V.J), therefore it must use a default configuration. In the case of the UART the default configuration is 57600 bits per second. If the communications configuration is different from this the programming tool will automatically switch baud rates after it commands reset into bootloader mode.

E. Bootloader communications over CAN

The bootloader has a default configuration for the CAN interface. The CAN interface runs at 1Mbit/s, uses short identifiers, forces the use of packet-over-CAN, and uses the identifier 0x400 for inputs and 0x401 for outputs. If the communications configuration is different from this the programming tool will automatically switch CAN configuration after it commands reset into bootloader mode.
XXVI. LOG FILES

Any time the display software is connected to IntelliJect (including simulated) it will continuously write two different log files. These files are written to the “~/Power4Flight/IntelliJect Display” directory and are named “efilog-date-time.csv” and “efitel-date-time.efitel”, where date and time refer to when the file was created. The efilog file contains human-readable comma-separated values stored in rows, suitable for opening with a spreadsheet or analysis package.

The efitel file records a binary record of the communications from IntelliJect. This can be used for data replay, and to regenerate the efilog file if needed. Note that there is no way to turn off the data logging. This is by design; the price of disk storage is dramatically less than the potential cost of lost data. The efilog file will normally record a line of information for every fast and slow telemetry frame from IntelliJect. However, if the engine is not running it will only record one line per second to keep the file sizes more tractable. If needed you can regenerate the full speed logging for a stopped engine by using the replay feature.

A. Logging toolbar

The number of lines written to the efilog file is shown in the logging toolbar at the top of the application (34,438 lines in Figure 87). You can also use this toolbar to create an averages log file. Each time you push the Avg button in the toolbar the software begins averaging all the data from IntelliJect. After the period given in the toolbar (5 seconds in Figure 87) the resulting average data are written to the averages log file.

![Figure 87: Logging toolbar.](image)

The averages log file has the same name as the normal log file; but ends in “-avg.csv” rather than “.csv”. The number of lines written to the averages file is displayed in the logging toolbar (2 in Figure 87). At the same time the average data is written the software will also write a standard deviations file, which gives the standard deviation of the data that was used to compute the averages. The standard deviations log file has the same name and number of lines as the averages log file, but the file name ends in “-dev.csv” rather than “-avg.csv”.

The logging toolbar also includes a commenting feature. Type a comment and hit enter to put the comment into log file. The comment is written to the last column of the log file. Each comment entered will be remembered in the comment drop-down, so you can select it again. If there is text in the comment when you push the Avg button the comment will be written into the averages and standard deviations log files.

B. Replaying an efitel file.

The efitel file can be opened and played using the menu option File->Open replay file... This will cause the display to update as it normally would if it were receiving packets of data from a live IntelliJect.
During replay you use the replay toolbar to pause or resume the replay, and to select the rate of replay. The status bar of the display software will indicate that you are offline, so you cannot send or request data. The toolbar displays the replay file position as a percentage.

Replaying an efitel file will cause the display software to generate a new efilog file. The name and location of a log file generated during replay is different from a live log file. The replay-generated log file will have the same location and the same root name as the “.efitel” file, but will start with “efireplaylog”. For example, when replaying the file “\temp\efitel-2017-10-11-08-21.efitel” the log file will be called “\temp\efireplaylog-2017-10-11-17-08-21.efitel”. If during the replay you use the **Avg** button the resulting files will have the same location and root name as the replay log file.
XXVII. EXTERNAL DEVICES

The display software works with several different external devices.

A. Cooling Fan

The software can control one or two cooling fans connected to either an Emerson Commander SK or Invertek Optidrive E2 or E3 variable frequency drive. The interface from the display software to the controller uses either modbus TCP (network) or modbus RTU (serial).

![Image of Cooling Fan Control](image)

Figure 89: Disconnected and connected command windows for the cooling fans.

Use the First and Second checkboxes to show or hide either set of cooling fan controls. At least one cooling fan is always visible.

1. Command

Push the Connect button to connect to the fan drive. The actual fan frequency is displayed in the center of the window. Stop will command the fan to zero speed, or the commanded fan frequency can be manually entered; or controlled with the analog input on the Emerson or the keypad on the Optidrive (Analog Control selected); or driven by engine head temperature feedback if Auto CHT is selected.

When using Auto CHT Target sets the desired head temperature for the feedback control. Notice that the first and second fan can have different target temperatures. The feedback signal for the first fan comes from CHT1; the second fan uses CHT2.

2. Control

The Control tab gives the gains used for feedback control of CHT. Both fans use the same gains.

- **Use IntelliJect output** should be selected to bypass the feedback control law and command the fan from the IntelliJect cooling telemetry. In that case the first fan is driven by the first cooling output and the second fan is driven by the second cooling output.

- **Kp** is the feedback gain from temperature error to desired rate of change of temperature.
• P is the proportional feedback gain from temperature rate error to fan frequency in tenths.
• I is the integral feedback gain from temperature rate error to fan frequency in tenths.
• F is the feedforward gain from throttle position to fan frequency in tenths.

3. Interface

On the interface tab you select if you talking to the Commander SK or Optidrive E2/E3 VFD.

• Select Commander SK to talk to the Emerson VFD.
• Select Optidrive E2/E3 to talk to the Optidrive VFD.
• Select Modbus TCP to use the network to talk to the VFD, otherwise you are using serial.
• IP specifies the internet protocol address of the VFD for use with modbus TCP.
• Port is used to select a serial port to work with modbus RTU. The ports are enumerated each time the interface tab is shown.
• Baud is the serial port interface rate in bits per second to work with modbus RTU.
• ID specifies the modbus address to use with the VFD.

4. Fan toolbar

A toolbar is available, see Figure 91. When connected the Fan indicator will be green and the actual fan frequency is displayed. The cylinder head temperature is also displayed on the toolbar. If both fans are selected in the fan control window toolbar includes information for both fans and head temperatures. Click on the Fan1 or Fan2 indicator to show the cooling fan window.
B. Horiba Gas

The software can connect to a left and a right Horiba Mexa 584L portable gas analyzers. The analyzers connect with a standard RS-232 serial port to the PC (not to IntelliJect). Figure 92 shows the window with one analyzer connected in different states.

The dropdown back at the top of the window is used to select the serial port on the PC that the analyzer is connected to. The indicator to the left of the port number will be green if the software is receiving data from the analyzer.

If the analyzer is warming up or zeroing the data will be “N/A”. The Error indicator will be red if the analyzer has an error, in which case hovering over the indicator will give a tool tip with the error description. Use the Stand By button to put the analyzer in stand-by mode. Use the Measure button to put the analyzer in measurement mode (which will start the air pump).

If the analyzer is the five-gas version with Oxygen the software will use the measurements to estimate the air fuel ratio and the trapping efficiency. The estimate is done using the methodology presented in SAE paper 901599 "AFR and Emissions Calculations for Two-Stroke Cycle Engines".

1. Horiba toolbar

A toolbar is available for Horiba summary, see Figure 93. When connected the Horiba indicator will be green and the measured CO is displayed. If you click on the Horiba indicator the Horiba Gas window will be shown.

C. Sound

The software can connect to an Extech digital sound meter, which provides a simple measurement of sound pressure (useful for comparing different exhaust systems). Figure 94 shows the sound meter window.

The dropdown at the top of the window selects the serial port on the PC to connect the sound meter. The indicator to the left of the port number will be green if the serial port is open.
1. Sound toolbar

A toolbar is available for sound meter summary, see Figure 95. When connected the \textit{Sound} indicator will be green and the measured sound pressure is displayed. If you click on the \textit{Sound} indicator the sound meter window will be shown.

![Figure 95: Sound meter toolbar, connected and disconnected.](image)

D. Fuel Pump / Meter

This window shows telemetry for the fuel system from the EFI, optionally for a Triplex fuel pump / meter, and optionally for a Max Machinery flow meter.

- \textbf{Reset Delivered} commands the ECU, Triplex, and Max meter to zero their respective delivered fuel estimates.

1. EFI shows telemetry from the ECU.
   - \textit{Pressure} is the fuel pressure in kPa.
   - \textit{Flow rate} is the fuel consumption rate in grams per minute; estimated by the ECU using injector characterization.
   - \textit{Delivered} is the estimate of total fuel in grams delivered by the ECU.

2. Triplex

The display software has support for the \textbf{Triplex fuel pump} from Currawong.

![Figure 96: Triplex and Max Machinery.](image)

If you are connected via CAN bus, this section of the window will be visible; allowing you to see telemetry from the pump and send pressure commands. Triplex maintains its own fuel pressure control loop. Use the \textit{Command} drop down to command the desired fuel pressure or turn the pump off.

- \textit{Connected} indicates if triplex data are visible on the CAN bus.
- \textit{Command} controls the pressure control loop on the Triplex.
- \textit{Flow rate} is the fuel delivery rate in grams per minute.
- \textit{Delivered} is the estimate of total fuel in grams delivered by the Triplex.
- \textit{Voltage} is the input voltage to the Triplex.
- \textit{Motor duty} is the duty cycle of the Triplex pump motor.
3. Max Machinery flow meter

The Max Machinery flow meter is a precision impeller-based volume flow meter that connects digitally to a counter timer. The display software uses channel 0 from a Measurement Computing counter timer module to count the edges from the meter. If the measurement computing driver software is not installed on the PC this section of the window will be hidden. The Max Machinery flow meter is only supported on Windows.

- Use the **Connect / Disconnect** button to toggle the connection to the counter timer. The connection will be made automatically when the software starts if the counter timer is connected to the computer. The light next to the button is green when connected.
- **Density** is fuel density in grams per cubic centimeter used to convert from volume to mass.
- **Calibration** gives the pulses per cubic centimeter of volume that flows through the meter.
- **Filter Time** sets the amount of time in seconds used to compute the flow rate from the meter measurement. The flow rate is determined the difference in the delivered fuel over the time interval.

4. Fuel Pump/Meter toolbar

The Triplex and Max have a toolbar which can be used to show key telemetry and commands.

![Figure 97: Fuel meter toolbar with Max and Triplex.](image)

E. NI Data Acquisition

The National Instruments (NI) data acquisition window is used to integrate measurements from tasks configured in the NI measurement and automation explorer. Any tasks which are present on the system can be selected in the drop down. Once selected the display software will query the NI software for new data 20 times a second, averaging any results and making them available as variables to be logged, charted, and displayed on the gauges dashboard. You can have up to 50 NI channels configured in a task.

![Figure 98: NI Window.](image)

The display software will remember any task you have selected and re-open that task when the software starts. This allows the software to learn the channel names before creating the log files, so the header line of the log file will include the channel names.
XXVIII. HARDWARE

A. Enclosure Dimensions

![Diagram of enclosure dimensions]

Figure 99: IntelliJect enclosure dimensions
B. Electrical Connections

IntelliJect has two main connectors. One notionally faces the engine, and the other faces the vehicle, so they are referred to as the Engine and Vehicle connectors. The pinouts were changed slightly for revision 3 of IntelliJect to provide more functionality. Revision 3 specific changes are in red; for earlier revisions these signals should not be connected.

1. Engine connector

The engine connector is Harwin part number M80-5302642, with mating connector M80-4662605. The connector has 26 pins in 2 rows of 13, as given in Table 3.

Table 3 Signals on the engine connector.

<table>
<thead>
<tr>
<th>PIN</th>
<th>NAME</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12V</td>
<td>OUTPUT</td>
<td>13.8V to injector 1</td>
</tr>
<tr>
<td>3</td>
<td>INJECTOR_1</td>
<td>INPUT</td>
<td>Primary low side injector driver</td>
</tr>
<tr>
<td>5</td>
<td>CDI_1_OUT</td>
<td>OUTPUT</td>
<td>6V Primary capacitive discharge ignition driver</td>
</tr>
<tr>
<td>7</td>
<td>6V</td>
<td>OUTPUT</td>
<td>CDI 1 power</td>
</tr>
<tr>
<td>9</td>
<td>GND</td>
<td>INPUT</td>
<td>CDI 1 return</td>
</tr>
<tr>
<td>11</td>
<td>CHT_1_SENSOR</td>
<td>OUTPUT</td>
<td>1.16 ma current source output for RTD/KTY8x Sensor</td>
</tr>
<tr>
<td>13</td>
<td>GND</td>
<td>INPUT</td>
<td>CHT 1 Return</td>
</tr>
<tr>
<td>15</td>
<td>MAT_SENSOR</td>
<td>OUTPUT</td>
<td>1.16 ma current source output for RTD/KTY8x Sensor</td>
</tr>
<tr>
<td>17</td>
<td>GND</td>
<td>INPUT</td>
<td>MAT return</td>
</tr>
<tr>
<td>19</td>
<td>THR_PWM_OUT</td>
<td>OUTPUT</td>
<td>Throttle servo signal</td>
</tr>
<tr>
<td>21</td>
<td>6V</td>
<td>OUTPUT</td>
<td>Throttle servo power</td>
</tr>
<tr>
<td>23</td>
<td>GND</td>
<td>INPUT</td>
<td>Throttle servo return</td>
</tr>
<tr>
<td>25</td>
<td>GND</td>
<td>INPUT</td>
<td>TPS return</td>
</tr>
<tr>
<td>2</td>
<td>TACH_1</td>
<td>INPUT</td>
<td>Hall sensor 1 input (5K internal pullup)</td>
</tr>
<tr>
<td>4</td>
<td>HEARTBEAT</td>
<td>OUTPUT</td>
<td>5V output: 1 ms pulse per crank revolution; alternate: spark3</td>
</tr>
<tr>
<td>6</td>
<td>5VA</td>
<td>OUTPUT</td>
<td>Hall sensor 1/2 power</td>
</tr>
<tr>
<td>8</td>
<td>GND</td>
<td>INPUT</td>
<td>Hall sensor 1/2 return</td>
</tr>
<tr>
<td>10</td>
<td>CHT_2_SENSOR</td>
<td>OUTPUT</td>
<td>1.16 ma current source output for RTD/KTY8x Sensor</td>
</tr>
<tr>
<td>12</td>
<td>GND</td>
<td>INPUT</td>
<td>CHT 2 return</td>
</tr>
<tr>
<td>14</td>
<td>TACH_2</td>
<td>INPUT</td>
<td>Hall sensor 2 input (5K internal pullup)</td>
</tr>
<tr>
<td>16</td>
<td>COWL2_PWM_OUT</td>
<td>OUTPUT</td>
<td>5V PWM output to drive cowl 2 servo.</td>
</tr>
<tr>
<td>18</td>
<td>CDI_2_OUT</td>
<td>OUTPUT</td>
<td>Secondary capacitive discharge ignition driver signal</td>
</tr>
<tr>
<td>20</td>
<td>6V</td>
<td>OUTPUT</td>
<td>CDI 2 power</td>
</tr>
<tr>
<td>22</td>
<td>CDI_2_GND</td>
<td>GND</td>
<td>CDI 2 return</td>
</tr>
<tr>
<td>24</td>
<td>5VA</td>
<td>PASSIVE</td>
<td>5VA for TPS sensor (alternate function for rev3: Injector 3)</td>
</tr>
<tr>
<td>26</td>
<td>TPS_AIN</td>
<td>PASSIVE</td>
<td>TPS analog input</td>
</tr>
</tbody>
</table>
Figure 100: Engine connector development harness: not recommended for flight.
2. Vehicle connector

The vehicle connector is Harwin part number M80-5302442, with mating connector M80-4132498. The connector has 24 pins in 2 rows of 12, as given in Table 4. This connector has main power, RS-232 serial, CAN, PWM throttle input, fuel pressure sensor input, fuel pump control output, the second injector, and the enable signal.

<table>
<thead>
<tr>
<th>PIN</th>
<th>NAME</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>THR_PWM_IN</td>
<td>INPUT</td>
<td>5V PWM input</td>
</tr>
<tr>
<td>3</td>
<td>GND</td>
<td>PASSIVE</td>
<td>PWM return</td>
</tr>
<tr>
<td>5</td>
<td>Fuel_Pressure_AIN</td>
<td>INPUT</td>
<td>0-5 volt analog input</td>
</tr>
<tr>
<td>7</td>
<td>5V</td>
<td>OUTPUT</td>
<td>+5VA power for fuel pressure sensor</td>
</tr>
<tr>
<td>9</td>
<td>GND</td>
<td>PASSIVE</td>
<td>Return for fuel pressure sensor</td>
</tr>
<tr>
<td>11</td>
<td>12V</td>
<td>OUTPUT</td>
<td>13.8V to injector 2</td>
</tr>
<tr>
<td>13</td>
<td>INJECTOR_2</td>
<td>INPUT</td>
<td>Secondary low side injector driver</td>
</tr>
<tr>
<td>15</td>
<td>RXD_RS232</td>
<td>INPUT</td>
<td>Main serial port receive data</td>
</tr>
<tr>
<td>17</td>
<td>TXD_RS232</td>
<td>OUTPUT</td>
<td>Main serial port transmit data</td>
</tr>
<tr>
<td>19</td>
<td>GND</td>
<td>PASSIVE</td>
<td>Main serial return</td>
</tr>
<tr>
<td>21</td>
<td>PUMP_SW_12V</td>
<td>OUTPUT</td>
<td>High side ON/OFF and PWM modulation for 13.8V fuel pump</td>
</tr>
<tr>
<td>23</td>
<td>PUMP_GND</td>
<td>PASSIVE</td>
<td>Return for pump</td>
</tr>
<tr>
<td>2</td>
<td>12V/24 Volt (nominal)</td>
<td>INPUT</td>
<td>8-28 Volts system input</td>
</tr>
<tr>
<td>4</td>
<td>GND</td>
<td>PASSIVE</td>
<td>System ground</td>
</tr>
<tr>
<td>6</td>
<td>SPARE_SENSOR</td>
<td>OUTPUT</td>
<td>1.16 mA current source output for RTD/KTY8x sensor</td>
</tr>
<tr>
<td>8</td>
<td>GND</td>
<td>PASSIVE</td>
<td>Return for ECU_EN, alternate return for spare sensor</td>
</tr>
<tr>
<td>10</td>
<td>ECU_EN</td>
<td>INPUT</td>
<td>ECU_EN active high enable 5V input (internal 5K pull down)</td>
</tr>
<tr>
<td>12</td>
<td>COWL1_PWM_OUT</td>
<td>OUTPUT</td>
<td>5V PWM output for cowl flap 1 Servo</td>
</tr>
<tr>
<td>14</td>
<td>6V</td>
<td>OUTPUT</td>
<td>Servo power</td>
</tr>
<tr>
<td>16</td>
<td>GND</td>
<td>PASSIVE</td>
<td>Servo return</td>
</tr>
<tr>
<td>18</td>
<td>PUMP_PWM_OUT</td>
<td>OUTPUT</td>
<td>5V PWM output for modulating brushless fuel pumps</td>
</tr>
<tr>
<td>20</td>
<td>GND</td>
<td>PASSIVE</td>
<td>PWM return</td>
</tr>
<tr>
<td>22</td>
<td>CANH</td>
<td>I/O</td>
<td>CAN high</td>
</tr>
<tr>
<td>24</td>
<td>CANL</td>
<td>I/O</td>
<td>CAN low</td>
</tr>
</tbody>
</table>
Figure 101: Vehicle connector development harness: not recommended for flight.
3. Temperature sensors

CHT1, CHT2, MAT, and spare temperature signals have a current drive of approximately 1.16 milli-Amps, suitable for some thermistors or resistance temperature detector (RTD). Software supports the KTY83 and KTY84 thermistors (which have a non-linear calibration curve), 1000Ω platinum RTDs, and any RTD that can be approximated with a linear calibration. The drive current will produce a voltage drop across the sense resistor that must be limited to 0 to 3.3V.

4. Power options

Ordinarily IntelliJect accepts 8 to 28 V input power; using an onboard regulator to make 13.8V for the injectors and fuel pump; and 6V for the servos and CDIs. Alternatively, the hardware can be configured to connect the input power directly to the injector rail, in which case the input power must not exceed 16V. Hardware can also be configured to connect input power directly to the servo and CDI power rail, rather than the 6V regulator. Contact Power4Flight for details of the configuration options.

5. Fuel pump control options

Typically fuel pump control is done through a high side switch (vehicle connector, pin 21) to the injector power rail. This is suitable for DC brushed pump motors. For pumps that have brushless motors the 5V output (vehicle connector, pin 18) should be used as the duty cycle control signal to the pump motor controller. The high side switch and the 5V output are both driven by the same control signal from the processor.

6. Third injector option

Pin 24 of the engine connector is normally 5V for an analog TPS sensor. However, hardware can be configured for a third injector low side switch on this pin.

7. Serial peripherals for the PC

Since PCs no longer have built in UARTs, serial connection to IntelliJect is typically accomplished with a USB to serial converter. Most converters should work out of the box, if the operating system has a driver for the device. We typically use the CommFront USB to RS-232 converter.

8. CAN peripherals for the PC

The IntelliJect display software gets its CAN hardware support from the Qt library. On windows this has support for the Systec USB-CAN module, and the Peak PCAN-USB. When using the systec module on Windows 10 you will need driver version 6.04 (or later). When using the PCAN-USB be sure to install the Basic API along with the driver.
XXIX. SOFTWARE RELEASE NOTES

The release notes provide the list of changes made for each released version of software, since version 1.0 of software was released on 13 November 2017.

A. Version 1.8.939 (6 December 2019)

1. New features for Intelliject firmware
   • Added new table: 'Manifold temperature'. This table provides a fuel multiplier which works like the head temperature table but depends on MAT. The charge temperature fuel multiplier now comes from either the manifold temperature table or the charge temperature table, depending on whether the manifold temperature table is enabled or not.
   • Added telemetry to specify if spark 1,2,3 and injector 1,2,3 are triggering from crank sense1 or 2 (or none at all).
   • Changed logic used to choose between crank sense 1 and 2. Previously each output (spark 1,2,3 and injector 1,2,3) would trigger off the nearest sensor, unless a sensor was in an error state. Now the determination of a sensor error looks at the age of the error and the sensor with the oldest error is used (unless the error is more than one minute in the past). This protects the system from using a sensor that is intermittently healthy.
   • Made crank sense 1 equal to crank sense 2: Previously crank sense 1 was always enabled and crank 2 was optionally enabled. Now both sensors have explicit disable configuration options. In addition, you can now specify which crank sensor is preferred if both sensors are healthy. If neither sensor is preferred, and both are healthy, then the old logic of using the nearest sensor is used.
   • Change dynamic errors reporting: previously an error would be cleared immediately upon resolution of the erroneous condition. Now the error is not cleared until after the slow telemetry output cycle. This guarantees that any error will appear at least once in the dynamic errors telemetry packet; even if the error sets and clears faster than the telemetry update rate.
   • Removed support for the old engine configuration packet. This packet was superseded by the new engine configuration packet in version 1.7. If you are upgrading from version 1.6 to 1.8, then load 1.7 first to correctly convert the engine configuration to the new format.
   • Removed center sense option from crank sense configuration.
   • Added test mode feature to suppress crank sense 1 or sense 2. If the crank sensor is suppressed it is still used to compute RPM and crank errors, but it will not trigger any spark or injector outputs. If the other sensor is working the spark and injector outputs will be scheduled on that sensor.
   • Added feature to measure and report in slow telemetry the crank shaft angle where crank sense 2 synchronizes. This feature can be used to determine if the crank sensor synchronization angles are set correctly with respect to each other.
• Added fuel multiplier table. This table provides a simple fuel multiplier and is intended to make engine fleet calibration easier by allowing engine specific fueling changes to reside in this table rather than the main fuel table.

• Added new test: user storage. The user storage test will exercise every bit in the user storage EEPROM. If the test passes the contents of the EEPROM will not be changed.

• Added new error source: user storage. Problems reading or writing user storage will set this error bit; and may generate a debug message.

• Added TPS source information to the slow telemetry, this complements the throttle command source in the fast telemetry.

• Added a new TPS source: the manifold pressure (via reverse lookup through the MAP estimate table). This TPS source was previously referred to as "Backup" and was only available if there was no TPS source and if IntelliJect was not driving the throttle. Now this source can be explicitly enabled in the configuration, even if the IntelliJect is driving the throttle. The MAP TPS source is the lowest priority TPS.

• Added a new TPS source: the CAN throttle reported position. The CAN throttle servo is now an option if IntelliJect is not driving the throttle: IntelliJect will not command the position or servo enable status if it is not driving the throttle. The CAN TPS source is lower priority than analog or PWM TPS.

• Added TPS error to the slow telemetry. The TPS error is computed when IntelliJect is driving the throttle and a TPS signal is available, otherwise the TPS error will be zero.

• Added option to run TPS feedback which modifies the throttle output so the measured TPS matches the command. This option is only available if IntelliJect is driving the throttle and a TPS is enabled.

• Added a second maximum throttle table, indexed by the hottest CHT and the density ratio. The actual maximum throttle used is the smaller of this table and the first maximum throttle table.

• Added a second minimum throttle table, indexed by the coldest CHT and the density ratio. The actual minimum throttle used is the larger of this table and the first minimum throttle table.

• Changed throttle rate limiter. Previously the rate limiter was applied to the output throttle; now it is applied to the throttle command input, and separately to the output of the RPM governor in direct control mode. This improves the performance of the RPM limiter.

• Added governor feature to enable a minimum and/or maximum RPM limiter. This feature is only available when the trajectory controller is enabled (i.e. trajectory gain is greater than zero). The limiter works when the governor is not running (i.e. throttle is commanded directly) to prevent the user from commanding a throttle that will cause the RPM to go below the minimum or above the maximum RPM configured in the governor.

• Expanded test mode to support tests for the third injector, spark, and cooling outputs.
• Added new telemetry message to report the third injector and spark operational details.

• Added software support for a third spark output. The third spark is enabled in the engine configuration and takes over the heartbeat output. Software support also includes a new spark3 delay table and a third spark on/off command.

• Added software support for a third injector (hardware support is available in IntelliJect rev3). This includes three new tables (ratio, phase, trim) for the third injector, and a third set of injector configuration options.

• Added injector option to specify how the spare temperature is used, either as a second MAT sensor, or as a third CHT sensor. This option is only used if independent injector operation is enabled. Since the spare temperature can now be a CHT sensor the spare temperature encoding in the sensors telemetry packet was changed to allow a higher range signal (same as the other CHT sensors).

2. New features for IntelliJect display

• Added automatic table text color; selecting between white or black text based on the background color of the table cell.

• Split the sdkexample.c module into two modules: sdkexample.c and sdkexamplesend.c. This facilitates using more of the SDK code in IntelliJect display.

• Added second configuration window. This is just a clone of the first window and provides a convenient way to view two different configuration categories at the same time (if you have enough display space).

• Added undo/redo capability. This works on a packet basis: any configuration packet which is sent automatically buffers a reverse packet which can be invoked with keyboard short cut or menu option.

• Enabled in-cell editing of values in the tables and throttle curve. In-cell editing is done by typing numbers when a cell is selected and IntelliJect is unlocked. The previous editing mode (double-click or enter to invoke the input dialog) is still available.

• Filtered the text display of RPM on the gauges display using a one second finite impulse filter. This affects only the text display, not the logged or charted data.

• Changed tick marks on the gauge display; major tick marks are now spaced every 1000 rpm and minor ticks are spaced every 250 rpm.

• Added open button to the configuration display (next to send and request) which can be used to open and send just that pages configuration data.

• Added logic to opening configuration files to prompt for confirmation if only the address information is being changed in the communications settings. Users typically want to change other communications settings without changing the addresses.

• Added 3D surface plot and heatmap for table visualization. These replace the previous contour plot visualization.
• Added the ability to convert a table from alpha-n to speed-density and vice versa. The conversion is only an option for tables that use throttle/load on the row axis and RPM on the column axis; and only if a MAP estimate table is populated.

• Consolidated the `About` and `Docs` menu into a single `Help` menu.

• Added support for dual cooling fans. If both fans are enabled the first cooling fan regulates CHT1 and the second fan regulates CHT2. If only one fan is enabled it regulates the hottest CHT.

• Added support for Optidrive cooling fan controlled over Modbus RTU.

• Added support for National Instruments data acquisition. The NI window can be used to load any task that was configured in the NI "Measurement and Automation Explorer" and acquire any channels specified by that task. NI channels are recorded to the replay file, logged, chartable, and can appear in the gauges dashboard.

• Changed throttle curve builder to add the option to define the desired relationship between input throttle command and RPM.

• Changed profile runner window to make better use of screen space by separating the chart and table with a tabbed layout. Also added right-click "goto" menu option to the chart and table.

• Added option to profile builder to use throttle and RPMs from the fuel table; or to split the throttle and RPMs from the table.

• Changed throttle and RPM command toolbars to use throttle and RPMs from the fuel table. Also added an option to apply the throttle curve in reverse, so the throttle commands yield output throttles rather than input throttles.

• Added table display "trails". The trail shows the history of the operating point on the table display.

• Added right click feature to strip charts to insert a comment or take average data, using a time which is older than the most recent telemetry. This makes it easy to mark interesting events after they have happened.

• Changed gas analyzer window to make better use of screen space by optionally hiding the left or right analyzer display.

• Updated gas analyzer window to compute AFR if possible. This uses the methodology presented in SAE paper 901599 "AFR and Emissions Calculations for Two-Stroke Cycle Engines".

B. Version 1.7.806 (3 December 2019)

1. Bug fixes for IntelliJect firmware
   • Adjusted temperature computation for MAT, CHT1, CHT2, and Spare temperature. The adjustment accounts for reverse voltage leakage current on the ESD diodes of the
temperature inputs. The adjustment is only active if the board major revision number is 3 or less and if the minor revision number is 4 or less (i.e. 2.5 and 3.5 boards are fixed in hardware).

- Adjusted temperature computation for MAT, CHT1, CHT2, and Spare temperature. The adjustment uses the CPU temperature to account for temperature induced variation in the current drive to the sensors.
- Fixed bug that could generate an erroneous crank wheel synchronization error if the engine was stopped and started.
- Changed initial state of the ignition outputs. Previously these were initially set active in anticipation of coil ignitions that need to be in the active state to prevent charging the coil. However, this does not make sense with the hardware inhibit of the ignition output that was added in rev2 hardware. Accordingly, the initial output state of the ignitions is now inactive.
- Improved default main and 12V current sensor calibration.

2. Bug fixes for IntelliJect display

- Improved crank sense timing chart x-axis resolution.

C. Version 1.7.805 (7 November 2019)

1. Bug fixes for IntelliJect firmware

- Fixed bug that could cause a floating point reset if a crank period was measured which was faster than the injector trim time.
- Fixed incorrect quick-restart crank sense configuration.
- Fixed incorrect CHT telemetry in the Piccolo ECU alternative protocol.
- Updated spare temperature telemetry: if spare temperature is being used as an MAT, and the spare temperature sensor has failed, the reported telemetry for the spare temperature will be the other MAT sensor (or the default MAT value if the primary has failed). This change does not affect injector operation, just telemetry.

2. Bug fixes for IntelliJect display

- Fixed bug that could prevent correctly saving or loading cooling configuration to a file if that file also contained "old engine" configuration data.

D. Version 1.7.803 (31 October 2019)

1. Bug fixes for IntelliJect firmware

- Fixed timing error that could produce occasional incorrect analog sensor readings. This would occur when receiving configuration change packets that would trigger an EEEPROM write.
• Improved crank sense decoding: using high tooth count wheels at high speeds in combination with high processor loading could cause problems in wheel decoding. The crank sense interrupt architecture has been changed to eliminate this possibility.

• Fixed bug that would always generate a crank "too late" error the first time the crank wheel synchronized.

• Fixed telemetry of MAT in the case where the spare temperature was configured as a second MAT. Previously this would result in MAT telemetry being reported as the average of the MAT and spare temp sensors. Now the MAT telemetry is just the value of the MAT sensor. This change only affects telemetry, not injector operation.

2. Bug fixes for IntelliJect display

• Fix for oscilloscope display, which was not correctly setting the time extents on the plot.

• Fix to correctly display the modified status for tables. This bug was introduced in 1.7.800 when dual tables were added.

• Fix to various display issues with the crank wheel timing window.

• Fix to automatically highlighting input text: sometimes the boxes could remain highlighted when they should not be.

• Fix to set the CAN addresses to default when running the software for the first time on a new machine.

• Fix to correctly suppress display of new windows when upgrading from older versions of software.

• Fix to correctly display the crank center sensing option in the engine configuration window.

• Deprecated crank center sensing and added warning message that the feature will be removed in the next version.

• Adding warning to the user to close the application after downloading a new version before installing the new version.

3. New features for IntelliJect display

• Added quick color feature to the table display: the six most recently used colors can be applied to the current table selection with a button.

E. Version 1.7.800 (13 October 2019)

1. Bug fixes for IntelliJect display

• Fix to correctly set the revcount when changing engine wear status.

• Fix to prevent changing table cell selection due to keystrokes when the table display has the focus.
2. New features for IntelliJect display
   - Added the option to show two tables at once on the table configuration window.

F. Version 1.7.799 (18 September 2019)

1. Bug fixes for IntelliJect firmware
   - Fix to prevent test mode from triggering spark or injectors when the system is not enabled.
   - Support for Currawong ECU protocol version 7.05 (firmware 6.40) which adds fuel pump rate control.

2. Bug fixes for IntelliJect display
   - Fix to throttle and rpm command bars to automatically highlight the command text when the control receives focus.
   - Fix for opening configuration file: if both new engine and cooling configuration are in the file do not initially select old engine configuration.

G. Version 1.7.798 (13 August 2019)

1. Bug fixes for IntelliJect firmware
   - Fix for test mode: Increased the timing precision of a test to 1 millisecond.

2. Bug fixes for IntelliJect display
   - Fix to saving IntelliJect configuration so old engine configuration (for version 1.6 and earlier) can be saved alongside the new engine configuration.
   - Updated the CAN database spreadsheet to be consistent with API 7 (firmware 1.7).
   - Fix for test mode: previously using the spark skip test would prevent other tests from working.

H. Version 1.7.797 (1 August 2019)

1. Bug fixes for IntelliJect firmware
   - Fix for receiving fuel pump configuration from the Currawong alternative protocol. The fuel pump bang-bang limit was not set correctly.

2. Bug fixes for IntelliJect display
   - Fix for sending engine wear data by loading a configuration file.
   - Fix to include spare temperature sensor data in the oscilloscope display.
   - Fixed storing automatic CAN IDs: when sending communication configuration changes the display software updates its connection settings to follow, and could forget the connection settings were using automatic IDs.
3. New features for IntelliJect display
   • Added the ability to check for newer versions of software from the Power4Flight website.

I. Version 1.7.796 (19 July 2019)
   1. Bug fixes for IntelliJect firmware
      • Fix to prevent user from simultaneously configuring throttle position sense and throttle command on the same input.

   2. Bug fixes for IntelliJect display
      • Fix for sending old engine configuration data (from .efi files saved from earlier versions).

J. Version 1.7.795 (9 July 2019)
   1. New features for IntelliJect firmware
      • Added heartbeat signal, which is a 1 ms output that is triggered on every crank synchronization. The IO signal is only available on rev 3 hardware.
      • Changed how the parameter mismatch detection is applied. Previously the mismatch check was only performed when IntelliJect booted up. Now the mismatch check is also performed anytime the configuration is changed.
      • Added new transient fuel table which only applies when the throttle rate is positive. The old transient fuel table functionality is changed so it only applies when the throttle rate is negative.
      • Added factory trim option for the temperature sensor current drivers. As part of this the default temperature current setting was changed from 1.144 mA to 1.16 mA to better match production hardware. This will slightly decrease the temperature reported from KTY8x and RTD sensors.
      • Added platinum RTD option to the sensor configuration for the temperature sensors. This is like the KTY8x options; and specifies that IntelliJect should use built-in conversion functions to translate the analog voltage to temperature.
      • Added misfire option to test mode. This can be used to force IntelliJect to skip 1, 2, or 3 consecutive spark outputs; allowing you to test how the system handles a misfire.
      • Added option for independent injector operation. This option changes the computation of the charge temp and head temp fuel multipliers. Normally these multipliers are the same for both injectors, but if the injectors are operating independently injector 1 will use CHT1 and MAT to determine its multipliers and injector 2 will use CHT2 and (optionally) spare temperature. The second set of fuel multipliers are now reported in previously unused space inside the sensors4 telemetry packet.
      • Updated default RPM controller gains. The new defaults are less aggressive.
• Removed coil ignition fault option. This feature was not used by anyone, and the fault detection will be unavailable in rev 3 hardware.

• Added options for SD card recording: Previously an SD card record would be started each time the system booted up. Now a record will only be started if the engine runs, unless the "always record" option is enabled. Also added the option to disable SD recording.

• Added the ability to specify SD card recording rates separately from telemetry rates.

• Removed the option to generate telemetry crank synchronously. This feature was rarely used and not worth the complication.

• Added option to receive telemetry from a generator control unit (GCU) on the CAN bus, in the format specified by the Northwest UAV GCU. This is the fourth alternative protocol, which can be combined with the other protocols. Reception of GCU telemetry triggers transmission (and SD recording) of a new native slow telemetry packet that sends the GCU data to the user.

• Added option to output CAN telemetry in the format specified by the Performance Electronics PE3 ECU. This is a third alternative protocol (in addition to the Currawong and Piccolo alternative protocols). The PE3 protocol is output only, and it can be combined with the Piccolo ECU protocol (but not the Currawong ECU protocol).

• Deprecated the load variable in the fast telemetry packet. This variable is still present in version 1.7, but it should not be used by client software, preferring instead to compute load from manifold and barometric pressure in the sensors packet. Later versions of firmware will use this field in the packet for new data.

• Added feature to retrieve a record of information from the SD card over a communications interface (CAN, UART, USB), rather than removing the card from IntelliJect.

• Improved the estimate of the rate of change of throttle used for transient fueling.

• Improved the estimate of the rate of change of RPM used for RPM control.

• Added expanded feedback control options for the cooling output. Previously cooling output was controlled with simple proportional-integral feedback. Now you can enable a trajectory controller, which can do a better job of intercepting a target temperature without overshoot.

• Added support for a second cowl flap output. The current IntelliJect hardware (rev 2 and earlier) only has one cowl flap PWM output, in which case the second output is only available for cowl flaps driven by CAN servo. Rev 3 IntelliJect hardware adds a second cowl flap output on PWM.

• Added option to split the cooling output logic. When this option is enabled, and both CHT sensors are healthy, CHT1 is used to control the first cooling output; and CHT2 is used to control the second cooling output.

• Added software support for spare temperature sensor. This sensor replaces the never used humidity sensor. The spare temperature sensor hardware is part of the original IntelliJect design but has not been enabled in software until now.
• Added fuel pump control option to set the pump output to maximum if the fuel pressure sensor is failed. If this option is not selected the pump output is driven by the feedforward gain if the sensor is failed.

• Added fuel pump control option to only enable the pump if the engine speed is non-zero. A configurable priming time is also added to allow the pump limited run time when the engine is not running for priming.

2. New features for IntelliJect display

• Added intake serial number to the acceptance test report.

• Added example code to the communications standard developers kit.

• Added new variables to stripcharts, csv log files, and the gauges window. The new variables are the GCU telemetry, injector 2 fuel multipliers, engine rotation direction, hobbs time, and error bitfields.

• Added the ability to specify the horizontal and vertical display compactness in the .ini file ("%Power4Flight%\IntelliJect Display.ini"). To override the automatic compactness add "CompactHorizontal=x" and/or "CompactVertical=x" where x specifies the pixel spacing between horizontal or vertical elements. These settings in the .ini file are overridden if you provide the command line options "-compachorizontal" and "-compactvertical".

• Changed installer to be version aware. This makes it much easier to have multiple versions installed simultaneously. By default the installer will place the application in \ProgramFiles\Power4Flight\IntelliJectX.Y, where X and Y are the major and minor version numbers. The uninstaller, start menu shortcuts, and desktop shortcuts are all now version aware.

• Adjusted error display to aggregate the TPS errors to a single error light and add the spare temperature sensor error light.

• Added table visualization display feature. This is a new window that provides line and contour plots of any table, as well as table smoothing. The feature is useful for seeing outlier points in the tables.

K. Version 1.6.742 (8 November 2019)

1. Bug fixes for IntelliJect firmware

• Fixed bug that could cause a floating point reset if a crank period was measured which was faster than the injector trim time.

• Fixed incorrect quick-restart crank sense configuration.

• Fixed incorrect CHT telemetry in the Piccolo ECU alternative protocol.

• Support for Currawong ECU protocol version 7.05 (firmware 6.40) which adds fuel pump rate control.
2. Bug fixes for IntelliJect display
   • Fix to correctly set the revcount when changing engine wear status.
   • Fix to prevent changing table cell selection due to keystrokes when the table display has the focus.
   • Fix for oscilloscope display, which was not correctly setting the time extents on the plot.
   • Fix to correctly display the crank center sensing option in the engine configuration window.

L. Version 1.6.741 (1 August 2019)

1. Bug fixes for IntelliJect firmware
   • Fix for receiving fuel pump configuration from the Currawong alternative protocol. The fuel pump bang-bang limit was not set correctly.
   • Fix to prevent user from simultaneously configuring throttle position sense and throttle command on the same input.

2. Bug fixes for IntelliJect display
   • Fix for sending engine wear data by loading a configuration file.
   • Increased number of decimal places displayed in the transient and injector ratio tables.
   • Fixed storing automatic CAN IDs: when sending communication configuration changes the display software updates its connection settings to follow, and could forget the connection settings were using automatic IDs.

M. Version 1.6.738 (9 July 2019)

1. Bug fixes for IntelliJect firmware and display
   • Fixed rounding error in packet encoding. Fields that were encoded as 16-bit floating point types (for example table data) were rounding towards zero rather than towards nearest.

N. Version 1.6.737 (19 June 2019)

1. Bug fixes for IntelliJect firmware
   • Fixed injector trim adjustment to phase: If the injector trim is positive the injector phase should be advanced by the trim time to account for the injector start delay; however, the phase was retarded rather than advanced.
O. Version 1.6.736 (11 June 2019)

1. Bug fixes for IntelliJect firmware
   - Fixed injector phase tables bug: if the table values crossed 0 degrees the interpolation could produce the wrong result. Values in the phase tables represent crank angles, and any math involving crank angles must be correctly wrapped, including the interpolation function. Before the fix interpolation between 350 and 10 degrees would produce a result near 180 degrees. After the fix the interpolation results in a (correct) value near 0 degrees.

P. Version 1.6.735 (21 May 2019)

1. Bug fixes for IntelliJect Display
   - Fixed bug which prevented saving the time range of charts if the chart that was changed was not the first one.
   - Fixed bug with table display of load operating point (as opposed to throttle operating point).
   - Fixed bug in acceptance test report that did not show all important RPM controller settings.
   - Fixed bug which would allow sending settings without first unlocking the configuration.
   - Fixed bug which could cause multiple modal dialogs to be open, making the application appear to be locked up, when in fact there was a modal dialog waiting for input that was covered by another dialog.

Q. Version 1.6.733 (25 April 2019)

1. Bug fixes for IntelliJect firmware
   - Injector 2 was using the minimum pulse setting from injector 1.

2. Bug fixes for IntelliJect Display
   - Fixed table display bug that would use the wrong CHT (if more than one CHT was enabled) to draw the operating point of the min throttle, max throttle, and spark retard tables.
   - Adjusted the result from the button "Set for 1000 Ohm Platinum RTD". The drive current used for the gain and offset calculation was changed to 1.16 mA to more closely match the real hardware.

R. Version 1.6.732 (3 April 2019)

1. Bug fixes for IntelliJect firmware
   - Fixed bug with determining rotation direction using two crank sensors. This would only affect engine operation if the "prevent reverse rotation" or "prevent normal rotation" option was selected.
   - Slight improvement to RPM derivative computation.
2. Bug fixes for IntelliJect Display
   • Adjusted max machinery fuel meter counter-timer interface to reject bounce signals. This fixes a flow meter calibration problem.

S. Version 1.6.729 (7 March 2019)

1. Bug fixes for IntelliJect firmware
   • Fixed bug on derivative computation for the RPM controller. This affects the trajectory controller more than the classical controller.

2. New features for IntelliJect display
   • Changed installer to be version aware. This makes it much easier to have multiple versions installed simultaneously. By default the installer will place the application in \ProgramFiles\Power4Flight\IntelliJectX.Y, where X and Y are the major and minor version numbers. The uninstaller, start menu shortcuts, and desktop shortcuts are all now version aware.

T. Version 1.6.725 (26 February 2019)

1. Bug fixes for IntelliJect firmware
   • Fixed bug that would prevent automatic sensor offsets from updating if the sensor reading was out of range, even if the out of range reading was due to an incorrect sensor offset.
   • Fixed bug that could cause automatic sensor offsets to be read incorrectly from saved parameters. This would result in subtle sensor offset changes which were typically not large enough to notice.
   • Fixed bug that could cause delay in reading of CAN servo data or alternative ID CAN data
   • Fixed bug that prevented PWM throttle command or PWM TPS from working correctly.

2. Bug fixes for IntelliJect display
   • Fix to remember and recall averaging period for logging averaging function. The average period can also be changed while average is running.

3. New features for IntelliJect firmware
   • Increased the maximum fuel pump PWM period from 4115 to 16403 microseconds. This allows slower pump PWM frequencies which may be helpful for pumps with larger inductance motors.
   • Added new formulation for charge temperature correction. A new parameter has been added to the engine configuration which sets the *CHT reference*; which represents the head temperature when the fuel table is determined. When the engine operates at a CHT equal to the reference value the charge temperature multiplier will be 1.0, no matter what value is in
the table. In all other respects the new formulation behaves the same as the old. If the reference CHT is set to 0 (as it will be if you upgrade from older software or read an old configuration file) the old formulation is explicitly used (for backwards compatibility).

- Added support for onboard data logging via SD card. The data logging is based on writing the normal communications traffic (including all telemetry) to the SD card. The SD card is journaled such that each time the EFI resets a new record of data is written to the card. The SD card must be formatted as FAT32 and then setup by the IntelliJect firmware. Once this is done the card can record continuously, overwriting the oldest data as needed. Data is retrieved by removing the SD card and plugging it into a PC, so that IntelliJect display can parse the files.

- Removed support for alternative fuel flow measurement based on external meter or pressure accumulator decay rate.

- Added cowl flap servo calibration to the test mode. This is the same as the current-based hard stop detection used for the throttle servo calibration.

- Augmented the servo calibration feature of test mode to specify the actual position of the servo hard stop, rather than assuming it is 0% or 100%.

- Added the ability to specify the PWM throttle input calibration separately from the output calibration. The PWM throttle input calibration applies to the PWM TPS and PWM throttle command. If the PWM input calibration is 0 the output calibration is used for the input.

- Added non-volatile reset count information to the reset report packet. This is in addition to the previous volatile reset count data. The nonvolatile reset count is incremented and placed in user storage each time IntelliJect boots up.

4. New features for IntelliJect display

- Added sensor configuration option to specify gain and offset appropriate for 1000Ohm platinum RTD temperature sensor. This is purely a UI feature that populates the gain and offset values appropriately.

- Added parsing tool for SD card data logging which converts the recording into one or more replay files that can be replayed in the display software.

- Added the ability to change the grid of text items in the gauges display. Right clicking on the label allows you to select a variable (from the save variable list as stripcharts). The resulting layout is automatically saved and can be explicitly saved and loaded as a display layout.

- Added the ability to specify user enable, ignition 1 enable, and ignition 2 enable in the profile runner. This makes it possible to implement a "mag check" in the checkout profile.

- Added the ability to highlight a range of cells in a table and interpolate across columns, or interpolate across rows, to change the values of the cells in the center of the selection.
U. Version 1.5 (1 February 2019)

1. Bug fixes for IntelliJect firmware
   • Disabled the USB interface. The USB stack is not as robust as it needs to be. It will be re-enabled in future firmware versions.
   • Fixed mistake in test mode packet that did not correctly handle tests with timeouts longer than 255 seconds.
   • Adjusted timing of digital barometer sensor to address occasional failed readings. The digital barometer is a backup sensor to the analog barometer.
   • Updated handling of table and sensor configuration packets to properly nack an index that is beyond the acceptable range. Previously if you tried to change or request a table or sensor configuration packet with an invalid index there would be no response. Now such a packet will generate a "Packet not recognized" nack response.

2. Bug fixes for IntelliJect display
   • Fixed infinite loop that could occur when attempting to open a replay file that did not use "efitel-" in the name.
   • Fixed a crash that could occur when attempting to build a throttle curve without any data.
   • Improved replay functionality, previously replaying files could give inconsistent replay rates if the display had been talking to a live IntelliJect.
   • Added request for all settings following firmware programming. Without this the version and settings display could be out of sync following firmware update.
   • Added request for all settings following firmware programming. Without this the version and settings display could be out of sync following firmware update.
   • Fixed table selection behavior. Previously selecting one or more cells in a tale and then changing those values would clear the selection when the updated table was returned from IntelliJect. Now the table selection does not change, making it easier to quickly change multiple table values.
   • Fixed sending settings when mixing new versions of the display with old versions of the firmware. Now the display will not send packets that it knows older IntelliJect firmware versions will not understand.

3. New features for IntelliJect (firmware and display):
   • Added support for toothed wheel crank angle sensors, which use one or more missing teeth to synchronize the position of the wheel. The crank wheel sensor provides improved timing for events spaced around the engine cycle, since the amount of delay time needed to schedule the event can be dramatically reduced. This feature includes support for wheels with a single gap, or wheels with two gaps of different sizes. Wheels with two gaps allow detection of engine rotation. As part of this feature the crank errors that can be reported
have changed, and the errors packet has been changed to accommodate the new crank sensor errors. As with the previous once-per-rev sensors there are two available sensors allowing redundant sensing. The dual crank sensors can each have their own wheel design, or one sensor can use a crank wheel sensor while the other uses a once-per-rev sensor.

- Added crank sense timing report and display. The timing report is generated on request and reports the time interval between successive crank sense events. This is useful for debugging crank wheels and examining engine speed variation through one or more cycles. The timing report can be triggered manually, continuously, or when a crank synchronization error occurs.

4. New features for IntelliJect firmware

- Changed cooling output to match throttle if the engine is not running. This makes it much easier to test and calibrate the cooling output.
- Added improved watchdogging. The hardware watchdog is now backed up with a software watchdog which makes it possible to determine where the code has faulted before performing the reset.
- Added maximum throttle for starting. If the engine is not yet running, and the throttle is greater than the maximum starting throttle, an error is reported, and crank sense events will not generate spark or injector outputs. The error will become sticky if a crank sense is detected in this state. The maximum throttle for starting is useful both as an operator/engine safety check, but also as a feature to clear a flooded engine by cranking it without fuel injected.
- Added safe mode to deal with bad settings data that could cause system resets. Safe mode is invoked if the system has reset due to errors more than 20 times (between power cycles). If that happens safe mode will be asserted, which prevents the system from loading any saved non-volatile settings data, thereby preventing whatever bad setting is causing the error. Once in safe mode the system settings data can be cleared if needed. The only recovery from safe mode is to reset to go back to normal operation mode.
- Added setting to center the active crank sense timing. This option tracks the time when the crank sense becomes active and inactive, using the center time to trigger the crank sense event.
- Added new test mode feature to inject crank sense errors on a running engine, either a skip or an extra tooth. The skip error causes the system to skip the processing of a single active crank event. The skip error injection applies equally well to once-per-rev or crank wheel sensors. For crank wheel sensors you can also insert a tooth, as though an extra tooth were erroneously detected. The errors injections can be performed on either crank sense 1 or crank sense 2.
- Added the ability to specify the synchronizing crank angle for reverse as well normal engine rotation.
- Added engine rotation direction sensing based on timing of dual crank sensors. This feature works even if neither crank sensor is a directional crank wheel. The only requirement is that both sensors are healthy and do not synchronize at the same (or 180 deg apart) crank position.
• Added the ability to prevent spark and injector outputs if the engine is rotating the wrong way.

• Added injector configuration option to specify the edge used for injector phasing. Previously the injector phase always gave the end of the injection event. Now it can be configured to specify the end (default), middle, or start of the injection event. The injector angle in telemetry gives the angle of the specified edge.

• Added new test mode feature for calibrating servo closed and open positions. This only applies to configurations where IntelliJect is driving the throttle servo. The test can only be started when the system is unlocked and disabled. The test drives the servo, monitoring the input current until it exceeds a threshold, then drives the servo back the other way until the current is reduced to 25 mA above the starting current. The resulting PWM is recorded as the closed or open throttle PWM (depending on which test was run).

• Added auto-offset feature for the main and 12V current measurements. This feature estimates the offset of the current measurement when the throttle is between 10% and 90% (i.e. off the stop) and IntelliJect is disabled (so the fuel pump, injectors, and ignition are not consuming current). In this case the expected current is very close to zero, so the auto-offset feature adjusts the sensor offset until the measured current is zero. As with the MAP sensor auto-offset, this feature must be enabled in the sensor configuration.

• Changed 2nd order barometric fuel multiplier table to use density ratio rather than barometric pressure ratio.

• Added a new table for density adjustment to the spark advance. The table value is multiplied against \((1.225 - \text{density})/1.225\) to compute a value that is added to the spark advance. Typically as the air density decreases the spark advance should increase, and this table provides the means to do that.

• Added injector term for fuel volume compensation based on the manifold temperature. The assumption is that the injector is at the same temperature as the manifold, and the fuel will reach the same temperature as it flows through the injector. The volume correction adjusts the injector opening time to compensate for changing fuel density. There is a new injector parameter that defines the fuel volume percentage change for degree C of the manifold temperature. The correction factor is 1.0 when the MAT is 15C.

V. Version 1.4.607 Update 1 (25 October 2018)

1. Bug fixes for IntelliJect firmware
   • Fixed occasional erroneous RPM reading on the first crank revolution.
   • Fixed problem with the digital barometer and OAT that could cause a watchdog reset if the sensor was enabled (by default it is not enabled).
2. Bug fixes for IntelliJect display:
   • Fixed drawing problem with the rpm gauge that happened if the gauges window was hidden when the application was started.
   • Fixed problem with chart signal selection when opening a display layout file which specified chart signals that were different than currently loaded.

W. Version 1.4.606 (18 October 2018)

1. Bug fixes for IntelliJect firmware
   • Changed the reporting of injector faults. Injector faults will not assert unless the engine is running, though they will remain asserted if the fault is present even if the engine stops. This is because an injector fault cannot be cleared except by performing an injection, and spurious faults could occur during engine setup.
   • Fixed bug in computation of temperature from the MS5637 sensor. This is a backup sensor and is not enabled by default.
   • Fixed incorrect trigger time in scope data if the scope was triggered after 4294967295 microseconds.
   • Fixed error reported if the fuel pressure was low while IntelliJect was disabled. Fuel pressure is expected to be low if IntelliJect is disabled.
   • Fixed temporary CAN communications bug that could occur when clearing all settings. This would happen when the settings would change to defaults, but the CAN masking hardware was not updated until the system reset. Now clearing all settings will trigger a re-initialization of the communications hardware.
   • Fixed bug that could cause a missed output (injection or spark) if multiple outputs were due at exactly the same time.
   • Fixed bug that caused an incorrect packet bit number in the nack response to a request packet which was requesting a packet not recognized by the firmware. This bug would lead to an infinite packet request cycle when a new configuration packet was added to the request list.
   • Fixed crank sense bug that could cause the system to think the engine was running when it was not. This would only occur if a single crank sense event had occurred in isolation, and the only consequence was that certain commands (such as test mode) would be locked out in this state.
   • Fixed fuel pump duty cycle reporting. Previously logic was used to handle the case where the communications output frequency was faster than the pump on/off frequency; the logic attempted to produce an average pump duty cycle. However this logic could make the output confusing in low duty cycle cases and has been removed. Now the pump duty cycle is simply the average duty cycle for the reporting period.
• Fixed computation of settings hash in the event that not all possible packets were part of the configuration. This would happen if the default settings for a particular category did not need to be changed. In that case the configuration hash would be different if the packet had been sent versus if it had not, even though the actual configuration was identical. Now any configuration packet which is not present in the configuration (factory or user) is automatically written to user storage at startup with default values, thereby guaranteeing that the packet will contribute to the configuration hash.

• Fixed computation of settings hash in the event that autocorrect was enabled for the MAP sensor. In that case the sensor offset would vary as needed in the settings, which would cause a spurious settings mismatch error. Now the autocorrect offset is stored separately from the normal sensor configuration data, so changes made by the auto-correct algorithm are not included in the configuration hash.

• Improved automatic MAP sensor offset correction. With the previous continuous algorithm it was possible to produce a small error when the engine was started, by changing the manifold pressure before IntelliJect knew the engine was running. The new algorithm is not continuous and will only update the offset 2 seconds after computing the new offset, if the engine is still not running.

2. Bug fixes for IntelliJect display:

• Improved setting the display layout to default. The previous method did a poor job of sizing the tabified windows.

• Fixed bug in charting of oscilloscope data. Scope data are now double-buffered and the display is only painted on the complete reception of the time window of scope data.

• Fixed bug that caused the last row of the throttle curve to be corrupted when deleting rows from the throttle curve.

• Fixed replay pause/play button which could get out of sync with the actual paused/playing status if you opened a second replay file.

3. New features for IntelliJect (firmware and display):

• Replaced the fuel pump output scope channel with an output edge error channel. This channel can be used to trigger the scope output to catch spark or injector outputs that have diabolical timing which cannot be satisfied.

• Increased the amount of engine wear data. In addition to hobbs time and engine revolutions IntelliJect now tracks the amount of time the engine was at high load, the amount of time the engine was running too hot, the peak engine temperature, and the number of times the engine was started. Hot time is subject to a time multiplier, such that increasing temperature causes hot time to accrue faster. Since engine wear is a regular telemetry packet, and adding the new data caused the packet to exceed the 8 byte data limit that is preferred for telemetry packets on the CAN bus, the engine wear packet now has a short form (with 8 data bytes) and a long form with all the data bytes. When the short form engine wear packet is used it is followed with a new packet that provides the extended engine wear data.
- Added logbook feature. The logbook consists of a series of packets that include name, date, and log entry description text. The logbook is stored on board flash of the IntelliJect processor (not the user storage eeprom). New log entries can be added when IntelliJect is locked but editing or erasing the log requires IntelliJect to be unlocked.

- Added maintenance feature. This consists of a new configuration packet which defines a maintenance schedule and a packet which defines the maintenance status; including an indicator if a maintenance item is due to be serviced, the serial number of the engine that the maintenance status applies to, and a command to indicate when the maintenance has been performed. Maintenance items in the schedule can be triggered by engine time, rev count, hot time, load time, peak CHT, or number of starts. The maintenance feature has two windows: a configuration window (and packet) that is controlled by the lock/unlock access; and a maintenance view window that displays the schedule and allows the user to indicate that maintenance service has been performed. Performing a maintenance item automatically adds an entry to the logbook.

4. New features for IntelliJect firmware
- Changed the behavior of the clear all storage command. Previously this command would preserve the engine wear information, now it clears everything, returning IntelliJect to a completely default condition.

- Changed default settings to include non-zero settings for the fuel table, spark table, starting fuel table, head temp table, and injector1 trim table. These defaults only apply if these tables are not present in the configuration; and are meant only to provide a useful starting point for a new configuration.

- Changed how IntelliJect saves volatile quick-restart data and user commands to reset proof storage. Previously this data was stored in SRAM, however the contents of SRAM are lost with a few hundreds of milliseconds after power is removed. Now IntelliJect stores this data in backup RAM on the processor. The backup RAM will remain valid for several seconds after power is removed, extending the interval of time that power can be interrupted while still performing a quick restart.

- Improved fuel pump control logic. You can now specify the PWM period used to control the pump motor, as well as specify a duty cycle rate limit to minimize current transients in the motor control.

5. New features for IntelliJect display
- Added menu option to toggle fullscreen mode for the display. Fullscreen mode hides the application title bar and covers the operating system taskbar.

- Added options to the acceptance test report to choose the files that were used to record the checkout data, the configuration file, and the checkout profile. These values are automatically populated but changing them is useful for cases where the acceptance test report is generated from a replay.
• On the CAN settings of the communication selection dialog, if you select an ID that is currently visible on the bus, the resulting value is transferred to the CAN input ID. Now it also transfers the value to the output ID by assuming that the output ID has the least significant set bit of the input ID cleared (above the type shift). This is not guaranteed to be the case but is likely to be so unless the CAN settings of the IntelliJect have been changed from their defaults.

• Added a button to the communication selection dialog to set the CAN configuration to default values. Also added support for auto-detecting the actual CAN IDs on the bus for cases where the CAN settings are at their default values, and then using those detected values.

• Improved the layout of individual windows - utilizing more consistent spacing and margins, and in some cases re-organizing the fields to better fit the dialogs.

• Added the ability to insert or delete throttle curve rows using a right click on the table.

• Changed the default display layout so that it centers the main window if the windows is not maximized or full-screened.

• Added support for dual Horiba gas analyzers (“Left” and “Right”). This is useful for twin engine development, in particular for doing cylinder balance. This features changes both the Horiba display and the toolbar, and also changes the layout of the log files.

6. Bootloader version 4:

• Version 4 of the bootloader adds support for reset commands in backup RAM, in addition to data in SRAM. This increases the amount of time power can be interrupted while still supporting quick restart. Reset commands in SRAM are still supported, so bootloader 4 can work with firmware versions older than 1.4

X. Version 1.3 (11 August 2018)

1. Bug fixes for IntelliJect firmware

• Fixed mistakes in emulation of Currawong ECU: the throttle curve was zeroes unless it was enabled, the RPM controller gains were scaled wrong by two orders of magnitude, and the charge temperature telemetry was computed wrong.

• Fixed native packet UART protocol handling that was broken when Currawong ECU emulation was enabled for the UART.

• Fixed incorrect sense on EFI enable signal.

• Fixed sensor sampling logic that would lose crank sync above 6000 RPM. This would lead to sensor value computation out of phase with the crank.

• Removed recursive packet search algorithm. This algorithm was not a bug, but it could cause substantial CPU blocking in the presence of communications noise (such as an unconnected UART cable).

• Fixed bug in measurement of CPU time, watchdog interval, and max interrupt blocking time. These values could previously be incorrect at high packet rates.
• Improved the logic used to fire the outputs (spark and injectors) for short periods of time, or for time events that are in the past. Previously these (rare) cases could cause the outputs to stall for 0.26 seconds, now they are handled gracefully.

• Fixed throttle logic for case where IntelliJect is driving the throttle output and is also measuring the throttle position through a TPS.

• Fixed reported injector duty cycle when engine was not running. If the engine is not running the injector duty cycle is reported as 0%.

• Fixed default gain on 12V current monitor

2. Bug fixes for IntelliJect display:

• Changed the color of the CHT readout in the head temperature gauge from red to black. The red text would become unreadable when the CHT became too hot and the gauge turned red.

• Fixed display of analog versus digital barometer reading on the sensors page, they were reversed.

• Improved the ability to work with configuration files when not connected to an IntelliJect.

3. New features for IntelliJect (firmware and display):

• Added support for alternate fuel flow rate measurement based on the pressure decay rate when the fuel pump is not running. This flow rate measurement can only be used with bellows-style fuel accumulators and dead-headed pumps with bang-bang control (i.e. Currawong single piston pumps). In that case the fuel pump will turn off for short periods and the pressure decay rate can be observed. This decay rate is directly related to the rate of fuel consumption, particularly when used with bellows-style fuel accumulators. The estimated flow rate is then integrated to provide an alternate fuel consumption estimate. The alternate fuel consumption is reported in its own packet, and can optionally be reported in the regular fuel telemetry packet.

• Added support for alternate fuel consumption measured by external counter (for example, the Max Machinery 210m). When enabled the throttle PWM digital input is used to count rising edges from the external meter (throttle PWM TPS and command must also be disabled). The alternate fuel consumption is reported in its own packet (the same packet as the pressure decay rate), and can optionally be reported in the regular fuel telemetry packet.

• Added the ability to set (or reset) the fuel used indicator. This is useful for cases where the fuel was topped up without the EFI being reset. It is also useful when doing flow rate calibrations.

• Changed alternative protocol handling to remove the option to specify shy mode over CAN. Instead all alternative protocols (UART or CAN) are implicitly in shy mode. Shy mode means that the native protocol outputs are suspended in favor of the alternative protocol outputs; although the native protocol outputs will resume on the first receipt of a native protocol packet. In addition, the display software now issues a request for data over the native protocol whenever a communications interface is opened, even before any telemetry is
received from the EFI. In this way the display software can successfully connect to an EFI that is configured for an alternative protocol.

- Removed option to change the active sense of the EFI enable signal. The active sense is fixed by the hardware (active high).

- Added option to specify a crank sense delay. The crank sense delay defines the amount of time that elapses between the assertion of the crank sense signal, and the propagation of that signal through the crank sense hardware filter.

- Added option for automatic manifold pressure offset correction. The manifold pressure sensor output has the potential to change over time as the sensor is exposed to fuel. When enabled this feature automatically adjusts the calibration of the MAP sensor so that, when the engine is not running, the MAP pressure reading matches the Baro pressure reading.

- Added option to specify manifold pressure error threshold in the engine configuration. If the manifold pressure from the MAP estimate table is different from the measured manifold pressure by more than the threshold the manifold pressure measurement is replaced with the estimate, and the associated error bit is set.

- Added TPS missing and TPS error bits to indicate if there is no valid TPS sensor when one is expected, or if the TPS sensor has a discrepancy with the throttle output.

- Added option to specify TPS error threshold in the throttle configuration. If IntelliJect is driving the throttle, and if a TPS sensor is enabled, but the difference between the throttle output and the TPS measurement is more than the threshold the TPS error bit will be set.

- Added new slow telemetry packet, TelemetrySensors4. This packet repeats the CHT and fuel pressure data from existing packets; but changes the encoded format to provide more information: the CHT signals can now go from \(-100^\circ\)C to \(+461.5^\circ\)C, and the fuel pressure is represented using floating point to accommodate negative values and more dynamic range. The equivalent signals from the previous telemetry packets are still available but should be considered deprecated. They will be removed from later versions of software.

4. New features for IntelliJect firmware

- Changed fuel pressure sensing for fuel pump control. Previously the fuel pressure used for fuel pump control was not subject to the low pass filter, now it is. Also updated the default fuel pressure sensor configuration to use a 5Hz low pass filter rather than 0.1 Hz.

- Added quick restart support. This feature saves key injector, spark, and throttle data to reset proof RAM, which allows IntelliJect to fire the spark and injector on the first crank sense after reboot (before the RPM can be measured or sensor data computed). In addition, the quick restart stores a flag to reset proof RAM that instruct the bootloader to skip the normal code verification step when the system reboots (only if the engine is running). Also re-ordered the bootup sequencing to minimize the amount of time required before the CPU can handle a crank sense event. The combination of these features makes it possible for IntelliJect to go through a reset with only a 100ms delay before resuming spark and injection outputs.
• Changed USB support: The USB peripheral is not configured or enabled until the VBus sensing signal is asserted, and only if the engine is not running at that point. This isolates key firmware functionality from any bugs in the USB software stack. It is not recommended to use the USB interface while an engine is running.

• Changed how the fuel pump duty cycle is reported in the slow telemetry. Previously bang-bang control of the fuel pump would produce a pump duty cycle telemetry which was not very meaningful. Now the reported pump duty cycle correctly represents the average pump cycle, even if the slow telemetry rate is faster than the rate at which the pump controller is cycling the pump on and off. This change only affects the reporting of the duty cycle, not the control applied to the pump.

• Changed how the min and max throttle tables are interpreted. Previously these provided the min and max input throttle. This has been changed so these tables provide the min and max output throttle.

• Changed how the RPM controller is run. Previously the controller was run once for each crank revolution; however, at higher engine speeds better performance can be achieved by running the controller slower to reduce noise. A configurable update rate (from 10 to 85 Hz, defaulting at 50Hz) has been added. The rpm controller now counts the time of successive revolutions until the elapsed time is longer than the desired update rate and then computes the RPM using the entire elapsed time to get a less noisy reading for the control law.

• Changed how the feedforward term of the RPM controller is implemented. The feedforward term was previously a straight line interpolation from the command RPM divided by the max RPM, multiplied by the feedforward gain. Now the feedforward term is the feedforward gain multiplied by the output of the throttle to RPM model of the RPM governor.

• Added feature to compute the manifold temperature from the US standard atmosphere (based on the baro pressure) if the MAT sensor is failed or disabled. The default value of the manifold temperature specifies the failed sensor reading at standard day pressure (101.325 kPa).

• Added feature to compute a backup TPS using the manifold pressure, barometric pressure, and MAP estimate table if the engine is running. The backup TPS is used if there is no other valid source of throttle position (efiDrivesThrottle is off, and analog or PWM TPS is invalid).

• Changed logic regarding low pass filter on the analog TPS. Previously this filter was disabled, and the throttle input filter was used instead. Now the filter is enabled, and the analog TPS value in telemetry will reflect the effect of this filter. The throttle input filter is still used, and its effect is visible in the throttle data.

5. New features for Intelliject display

• Increased the resolution of the fuel used display. The display now gives fuel consumption to tenth of a gram. This much resolution is useful when doing injector flow testing over short time intervals.

• Added the ability to control the line width in strip charting.
• Added the fuel used multiplier to the acceptance test report.

• Added a feature for scoring an engine checkout. The scores are based on the rpm, smoothness, overshoot, lag, and transition whenever the throttle is moved during the checkout (assuming a fixed pitch propeller load).

• Added more information to the reset report dialog, for resets caused by exception.

• Added telemetry file recording of packets sent by the display to IntelliJect (packets from IntelliJect have always been recorded). These packets are written to the same file as the packets that are received from the EFI. To determine the direction of the packet (to or from the EFI) the new packets are written to the file in a modified form that uses a different synchronization 1 byte. Such a packet is invalid in any context other than the telemetry file. When the display replays a telemetry file the special packets are used to populate the packet log window, indicating packets that went to the EFI when the file was recorded.

6. Bootloader version 3

• Version 3 of the bootloader adds support for quick restart. Quick restart is enabled when the firmware writes a special flag to reset proof RAM. When version 3 of the bootloader detects this flag it will skip the normal code verification check, so that the firmware can start as quickly as possible and keep the engine running.

Y. Version 1.2 (4 April 2018)

1. Bug fixes for IntelliJect firmware

• Changed the fuel pump switch frequency to 500Hz from 1kHz to better match the turn on times of the switch hardware.

• Fixed problem with ignition maximum dwell time - which would occasionally cause erroneous spark timing.

• Fixed application of injector trim time. Previously the trim was applied to the end of the injection pulse. Now the trim is applied to beginning of the injection pulse if it is positive, and to the end of the injection pulse if it is negative. This should better line up the actual fuel injection with the desired injector phase.

• Fixed ignition output sense to be the correct polarity. Now if you select the engine configuration option “Ignition Active High” the ignition output will be active high.

2. Bug fixes for IntelliJect display:

• Changed replay processing so that a bad or unrecognized packet does not pause replay.

• Added missing documentation to the packet log display for fictitious packets (from non-IntelliJect devices).

• Improved handling of floating displays when the display layout is locked.
3. New features for IntelliJect (firmware and display):
   - Added high speed oscilloscope feature. The oscilloscope records the analog signals at their base sample rate (1kHz) and digital signals asynchronously to generate a triggerable high speed display of data with microsecond resolution. This display can be used to debug key timing events, and to study the analog signals intra-cycle. The scope setup is volatile and not considered part of the EFI configuration, although the setup can be requested along with the normal configuration information.
   - Added debug message. This message is used by firmware developers to insert human readable debug strings where needed. The debug information is displayed in the packet log.
   - Added support for the Currawong Engineering CAN servo. IntelliJect can be configured to command a CAN servo for throttle and cowl flap (cooling) control. IntelliJect will report an error bit if a CAN servo is expected but not discovered, or if the servo is reporting an error code.
   - Added test mode feature. Test mode is used for hardware debugging and test. Test mode can simulate a crank signal, run injectors, ignitions, and fuel pump options. Test mode is only available if the engine is not running; and is automatically canceled if the engine starts running.
   - Added the ability to perform calibration of sensor offset or gain by providing the true sensor reading and computing the gain or offset that makes the raw data match the provided sensor output.

4. New features for IntelliJect firmware
   - Changed interpretation of injector phase table. The injector phase table now gives the crank angle when the injection ends, rather than the middle of the injection.
   - Augmented the interrupt time value in the CPU telemetry packet. Previously this was the worst-case amount of time spent in any one interrupt. Now it is that OR the maximum amount of time that interrupts were blocked, whichever is greater.
   - Added MMFAR and BFAR registers to the exception reset report packet. These registers report the address of an invalid memory access (if any).
   - Added new crank sense error “Missed crank sense edge” for crank sense 1 and 2. A missed edge is asserted when the crank sense interrupts for two consecutive edges of the same type (active or non-active). A missed edge error is distinct from a crank late or crank bounce error.
   - Added feature to track the time between active and non-active crank sense edges. This time is used to reconstruct the active edge time if the active crank sense edge is missed.

5. New features for IntelliJect display:
   - Changed the name of the temperature difference table. It is now called the “charge temperature” table. The functionality of the table has not changed.
• Added acceptance test report feature to automatically generates a report of the IntelliJect configuration and the charted performance.

• Added the ability to un-synchronize a chart, so that its time axis does not follow other charts. This makes it possible to have a chart scrolling live data while having another chart reviewing old data.

• Changed handling of nack messages to suppress the nack warning message while in replay.

• Added display of replay file progress (as a percentage) in the replay toolbar

• Separated the error display into two windows, one for dynamic errors (errors asserted now) and one for sticky errors (errors asserted in the past).

Z. Version 1.1 (5 February 2018)

1. Bug fixes for IntelliJect firmware

• Improved interrupt time tracking. Not all interrupts were contributing to the total interrupt time counter.

• Improved processing of CAN transactions by enabling CAN masking.

• Fixed handling of interrupts that could cause missed crank sense events.

• Changed the logic for the crank sense active high option; to account for the crank sense inversion on the hardware. “Active high” now refers to the signal at the IntelliJect harness, not the CPU.

• UART baud rate configuration is now being correctly applied at bootup.

• Fixed analog sensor sampling to prevent sensor overruns, which degraded the sensor accuracy.

• Changed the process of storing configuration data in the user storage space so that IntelliJect remains responsive while the storage operation is proceeding.

2. Bug fixes for IntelliJect display:

• Fixed bug that was preventing the strip chart auto scale setting from saving and restoring correctly.

3. New features for IntelliJect (firmware and display):

• Added feature to select load -or- throttle on a table by table basis rather than globally in the engine configuration data.

• Added option to force a sensor to default configuration.

• Added support for KTY83 temperature sensor.

• Added option to force communications configuration to default.
• Added new communications options, including the option to support the Currawong ECU protocol and the Piccolo ECU protocol.

• Added telemetry for communications system performance, which provides error indicators for USB, UART, and CAN systems.

4. New features for IntelliJect display:

• General improvements to layout and usability.

• Improvements to windows installer.

• Switched software to 64-bit on Windows (macOs and Linux were already 64-bit).

• Added display of table output. This makes it easier to see the exact value that a table is computing, rather than trying to eyeball it based on the operational condition.

• Added comment feature. The comment feature allows you to enter a comment when something interesting happens. The comment is saved in the replay .tel file, and is written to the log file, and is displayed on the strip charts.

• Rebranding from generic “Power4Flight EFI” to “IntelliJect”.

• Added ability to save and open configuration data in human readable text format.

• Added configuration comparison feature; you can compare IntelliJect configurations given by two files, or by a single file and a live IntelliJect.

• Added save and open configuration selection option. This makes it easy to pick and choose what pieces of the configuration should be updated from a file.

• Added alarm for CHT sensor failure.

• Added support for firmware update over CAN.

• Added feature to show what CAN IDs are active on the bus. This is helpful to the user when they are trying to discover the ID of the IntelliJect.

• Added preliminary support for CAN bus devices other than the Systec USB-CAN module.
### Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha-N</td>
<td>A type of fuel injection algorithm which is primarily based on the throttle position (alpha) and the engine speed (n).</td>
</tr>
<tr>
<td>Bang-Bang</td>
<td>A type of feedback control in which the control output is on or off, rather than continuously varied.</td>
</tr>
<tr>
<td>BDC</td>
<td>Bottom Dead Center, the crankshaft position with the piston as far from the head as possible.</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller Area Network, an automotive communications bus.</td>
</tr>
<tr>
<td>CDI</td>
<td>Capacitive Discharge Ignition, a type of spark plug ignition which uses capacitors to deliver the spark energy.</td>
</tr>
<tr>
<td>CHT</td>
<td>Cylinder Head Temperature, the temperature of the metal that makes up the cylinder.</td>
</tr>
<tr>
<td>CPU</td>
<td>Central processing unit, the main processor of Intelliject.</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check, a method of computing a unique description of a hunk of data. The CRC can later be recomputed to verify the data have not changed.</td>
</tr>
<tr>
<td>EFI</td>
<td>Electronic Fuel Injection.</td>
</tr>
<tr>
<td>FET</td>
<td>Field effect transistor, an electrical switch which can be controlled by an IO line and connects or disconnects power to a load.</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit.</td>
</tr>
<tr>
<td>ICD</td>
<td>Interface Control Document.</td>
</tr>
<tr>
<td>IO line</td>
<td>Input or output signal on a processor.</td>
</tr>
<tr>
<td>IP</td>
<td>Internet protocol, the packet protocol used for basic communication over the internet and local networks.</td>
</tr>
<tr>
<td>MAP</td>
<td>Manifold Air Pressure, the air pressure measured downstream of the throttle butterfly, and upstream of the intake reed.</td>
</tr>
<tr>
<td>MAT</td>
<td>Manifold Air Temperature, the air temperature measured inside the throttle body.</td>
</tr>
<tr>
<td>Nack</td>
<td>Not-Acknowledge, a message from Intelliject indicating that it cannot perform a command.</td>
</tr>
<tr>
<td>PC</td>
<td>Personal computer, running windows, macOS, or linux. Used for the Intelliject display software.</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse Width Modulation, a means of signaling a proportional signal based on the amount of time a signal is high, typically used to control servos.</td>
</tr>
<tr>
<td>RS-232</td>
<td>Electrical signaling standard for UARTs.</td>
</tr>
<tr>
<td>RTD</td>
<td>Resistance temperature detector. Temperature is sensed by driving a fixed current through the resistor and measuring the resultant voltage drop.</td>
</tr>
<tr>
<td>SDK</td>
<td>Standard Developers Kit, a package of documentation and software source code to help developers write software that works with Intelliject.</td>
</tr>
<tr>
<td>Speed-density</td>
<td>A type of fuel injection algorithm which is primarily based on the manifold pressure, and the engine speed.</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission control protocol, a high level protocol for transporting data over IP.</td>
</tr>
<tr>
<td>TCR</td>
<td>Temperature coefficient of resistance for an RTD, given in Ω/Ω°C.</td>
</tr>
<tr>
<td>TDC</td>
<td>Top Dead Center, the crankshaft position with the piston as close to the head as possible.</td>
</tr>
<tr>
<td>TPS</td>
<td>Throttle Position Sensor, a sensor that measures the position of the throttle butterfly, typically an analog potentiometer.</td>
</tr>
<tr>
<td>UART</td>
<td>Universal asynchronous receiver transmitter, the basic hardware of a serial port.</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus.</td>
</tr>
</tbody>
</table>