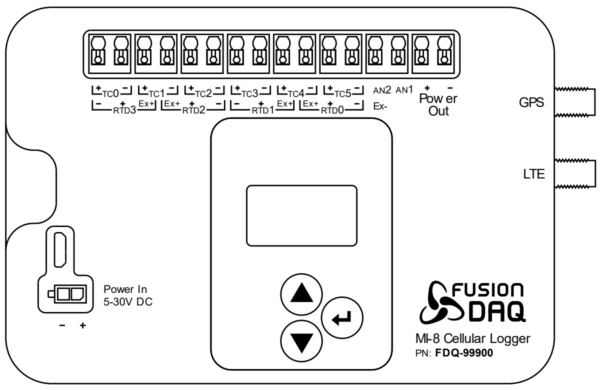
FusionDAQ.com

MI-8 User Manual

8-Channel, Mixed Input, Cellular Data Acquisition System



Part Number: FDQ-99900

Table of Contents

Revision History

Date	Author	Notes
2/12/2022	J. Leonard	Initial release
8/27/2023	J. Leonard	Improved config.json documentation. Updated triggering options and configuration.

Table 1 - Revision History

Glossary

ADC	Analog to digital converter
CAT-M	LTE network protocol intended for low power sensors as part of the Internet of Things
CSV	Comma separated value – Text based file format to encode data in a 2D table
DAQ	Data Acquisition unit
GNSS	Global Navigation Satellite System – Generic term describing any satellite constellation that provides positioning data
GPS	Global Positioning System – Often used as a generic term equivalent to GNSS to describe satellite based navigational systems
HTTP	Hypertext Transfer Protocol – Network protocol commonly used to interact with web servers
ICCID	Integrated Circuit Card Identification Number – Unique 18-22 digit code that includes a SIM card's country, home network, and identification number.
IMEI	International Mobile Equipment Identity – Unique ID for a cellular modem/phone
IOT-NB	LTE network protocol intended for low power sensors as part of the Internet of Things
JSON	Open standard file format used to describe data in a human readable format
LNA	Low Noise Amplifier – Signal amplifier, traditionally used in the context of radios and antennas
LTE	Long-Term Evolution – Fourth generation (4G) wireless standard
MI-8	8-Channel, Mixed Input DAQ
MQTT	Lightweight, publish-subscribe, network protocol
NMEA	National Marine Electronics Association
OEM	Original Equipment Manufacturer
RSSI	Received Signal Strength Indicator
RTD	Resistance Temperature Detectors - Sensors used to measure temperature
SIM	Subscriber Identification Module – Integrated circuit (IC) used to identify and authenticate subscribers on cellular networks

Table 2 - Glossary

Introduction

The MI-8 is a compact, 24-bit data acquisition system (DAQ), built for years of reliable operation in any environment. It is designed to measure up to six thermocouples or up to four resistive inputs such as RTD temperature sensors and strain gauges. Measurements may be stored locally on a mini-SD card, or may be pushed to a remote server via the integrated cellular modem.

With its wide supply voltage support and extended operating temperature range, the MI-8 is ideal for remote, battery powered and outdoor installations.

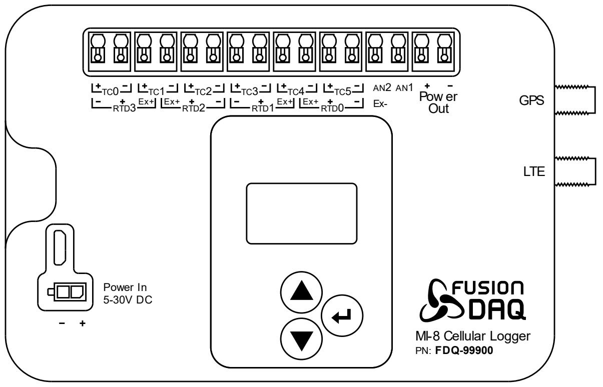


Figure 1 - With Enclosure

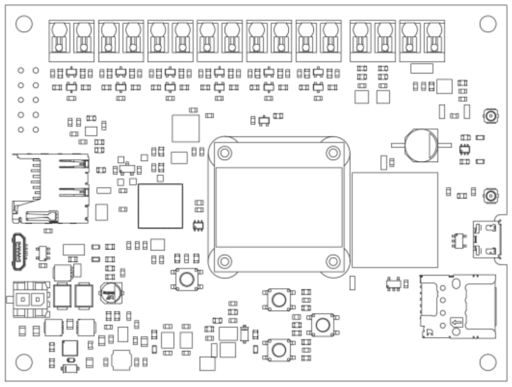


Figure 2 - OEM Configuration

The MI-8 is available with and without an enclosure to support custom packaging solutions. For example, the enclosure-less (OEM) configuration is often mounted into a IP-67 rated NEMA enclosure.

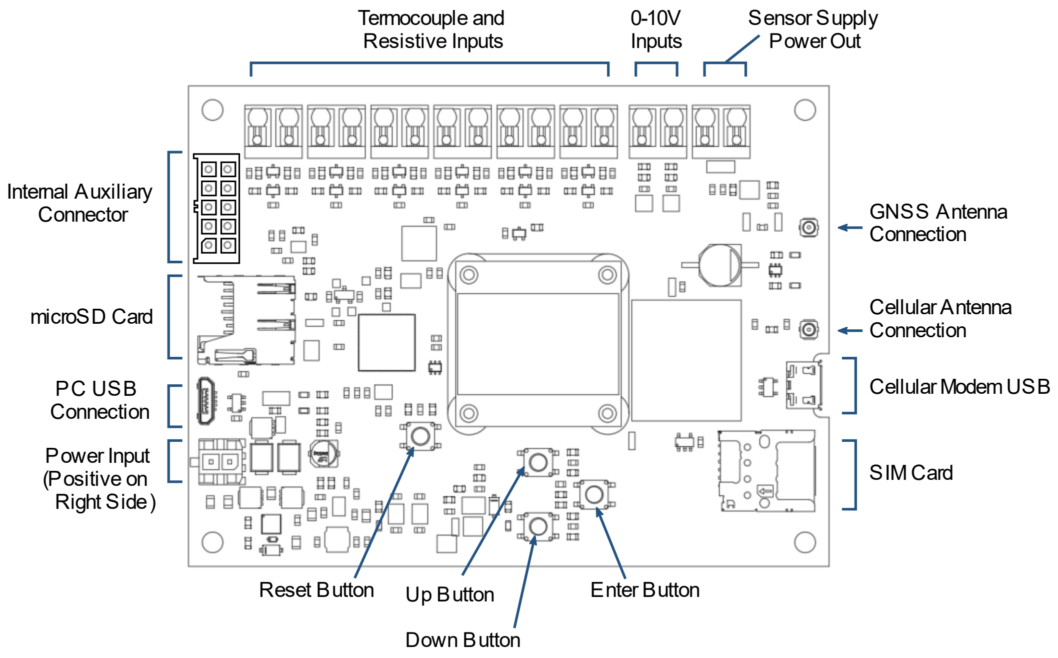


Figure 3 – Circuit Board Layout

Multiple power options are available. The MI-8 may be powered by 5V DC through the microUSB connector (PC USB Connection) on the left side of the circuit board. Alternatively, 5 to 30VDC may be supplied using the adjacent 2-pin Molex mini-fit connector (Power Input).   This wide range allows the MI-8 to be powered by existing 12V and 24V supplies common in industrial settings, or directly by a battery.   This power input is also tolerant to brief 40V voltage spikes and may be connected directly to the 12V system in automotive applications.

The two right-most pins along the top edge of the MI-8 circuit board provide regulated 14.5V power to supply external sensors. This voltage remains constant, even as the MI-8 input voltage changes.   This provides a convenient method for powering sensors in a battery powered installation. The regulated supply output voltage may be customized at the time of order. Common voltages are 5V and 12V.

The primary analog inputs are arranged at the top of the board. Thermocouple and resistive inputs utilize the twelve left-most pins. Two additional 0-10 Volt analog inputs are located to the right. Additional analog and digital inputs are available directly on the circuit board through the internal auxiliary connector (top left corner of circuit board).

The microUSB connector (Cellular Modem USB) on the right side of the circuit board, adjacent to the SIM card, is used to update the cellular modem firmware. This connection should not be used in normal operation.

GNSS (GPS) and cellular antenna connections are located on the right side of the circuit board. Male U.FL connectors are provided on the circuit board.

Limits and Specifications

Name	Description		Min	Nominal	Max	Unit
V_{Supply}	Supply Voltage		4.8	-	30	V
$T_{Ambient}$	Ambient Temperature		-40	-	85	C
I_{Supply}	Supply Current	V_{Supply} = 5.0 V V_{Supply} = 12.0 V V_{Supply} = 24.0 V				mA
P_{Supply}	Supply Power					mW
$P_{Standby}$	Power consumption while in standby					mW
$V_{CommonMode}$	Common mode voltage which may be applied to DAQ analog input pins		-2.4	-	2.4	V
V_{Sense}	Maximum voltage which may be measured by analog differential DAQ input pins		-4.8	-	4.8	V
$V_{SenseLimit}$	Maximum voltage which may be applied to DAQ differential analog inputs without causing physical damage		-6.4		6.4	V
V_{SDCard}	SD Card supply voltage			3.3		V
V_{SIM}	SIM card supply voltage		1.75	1.8	1.95	V
V_{LNA}	GPS active antenna (LNA) supply voltage			3.8		V
Z_{GPS}	GPS Antenna Impedance			50		Ω
$Z_{Cellular}$	Cellular Antenna Impedance			50		Ω
$I_{SenseLimit}$			-15		15	mA
R_{Sense}	Resistance input measurement range		0.12	-	4000	Ω
R_{Input}	Analog input equivalent series resistance		489	499	509	Ω
V_{Aux}	Voltage limits which may be applied to the digital and analog inputs on the internal auxiliary connector		0	-	3.3	V
$I_{AuxLoad}$	Maximum current draw form internal auxiliary connector digital outputs		0	-	15	mA
$I_{AuxSupply}$	Maximum current draw from auxiliary connector 3.3V supply		0	-	100	mA
$V_{PowerOut}$	Voltage of external sensor power supply		13.9	14.5	15.0	V
$I_{PowerOut}$	Maximum current draw from external sensor power supply		0	-	150	mA
t_{sample}	Minimum sample period		100	-	-	ms

Table 3 – Limits and Specifications

Operating Instructions

User Interface

The MI-8 includes three push buttons (Up, Down, and Enter). The up and down buttons are used to navigate between the various display pages. Each display page contains up to three MI-8 measurement channel values. The exact number of visible channels is based on the configuration parameters in the config.json configuration file on the SD card. Only enabled measurement channels will be visible. A scroll bar on the right side of the display indicates the relative position of the currently displayed page.

The last (bottom) display page may be used to eject the memory card. The memory card should always be ejected prior to removal to ensure data logs are properly closed and data loss does not occur.

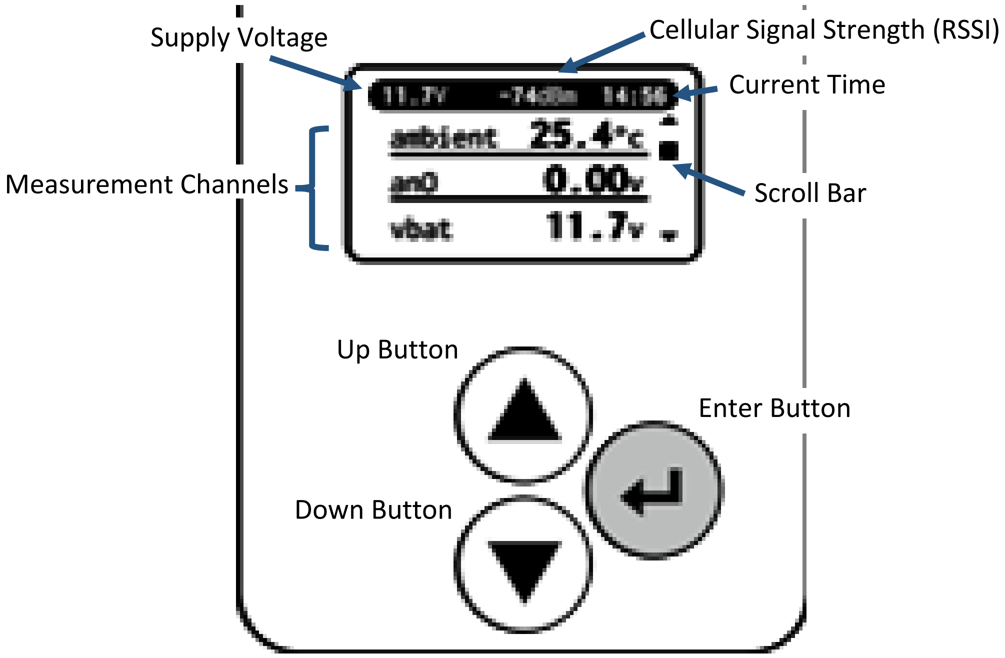


Figure 4 – MI-8 User Interface

The supply voltage (displayed at the top of all measurement pages) can be used to estimate the state of charge in battery powered installations.

The 'rssi' measurement signal may be used as an indication of signal quality. RSSI is measured in dBm and typical values fall between -110 and -54 dBm. Higher numbers indicate a stronger signal. Note that RSSI is a negative number, so numerically higher values (better signal quality) are closest to zero.

Configuration

The MI-8 is configured through a JSON formatted text file in the root directory of the SD card named "config.json". This file is made up of configuration parameters and their values. It may be modified using any common text editor.

Configuration parameters are *italicized* in the remainder of this document. In general, the MI-8 must be power cycled (powered off and then back on) after the configuration file is modified in order for changes to take effect.

Cellular Network Communication

The MI-8 supports LTE CAT-M1 and CAT-NB1/NB2 cellular communication.

Specification	**CAT-M1**	**CAT-NB1/NB2**
Supported Bands	1, 2, 3, 4, 5, 8, 12, 13, **14**, 18, 19, 20, 25, 26, **27**, 28, 66, 85	1, 2, 3, 4, 5, 8, 12, 13, 18, 19, 20, 25, 26, 28, 66, **71**, 85
Downlink	300 kbps	20 kbps
Uplink	300 kbps	10.3 kbps
Modulation Type	QPSK / 16QAM	BPSK / QPSK

Table 4 – Cellular Modem Specifications

Many cellular networks require an access point name (APN) in order to connect to their network. The APN is specified through the *network.apn* configuration parameter. If a username or password is required, they may be specified using the *network.username* and *network.password* configuration parameters. All three parameters are optional. Set any unused value to an empty string (two quotation marks with nothing in between) if they are not needed by the cellular network.

GNSS Satellite Based Position

The MI-8 combines data from the GPS, GLONASS, Galileo, and BeiDou Global Navigation Satellite Systems (GNSS) to determine position. An initial position fix is generally available within 30 seconds of power up when the antenna has a clear view of the sky. The initial fix may require several minutes if the antenna does not receive a high-quality signal.

Specification	**Value**
Tracking Sensitivity	-159 dBm (GPS) / -156 dBm (GLONASS)
Cold-start sensitivity	-148.5 dBm
Accuracy (Open Sky)	0.74 m (CEP50)
Receiver Type	16-channel, C/A Code
GNSS L1 Frequency	1575.42 ±1.023 MHz
GLONASS Frequency	1597.5 – 1605.8 MHz
BeiDou Frequency	1559.05 – 1563.14 MHz
Galileo L1 Frequency	1575.42 ±1.023 MHz

Table 5 – GNSS Receiver Specifications

Due to hardware design, the cellular modem and GNSS receiver may not be used at the same time. The cellular connection is interrupted any time the GNSS receiver is active and the GNSS receiver is unable to resolve position when the cellular modem is active. The MI-8 will automatically switch between the two radios each time a new position fix or network notification is required. The interval between GNSS position measurements may be adjusted using the *gnss_period* configuration parameter. In general, the GNSS sample interval should be a long as possible to minimize cellular connection interruptions and to minimize power consumption.

Date and Time Determination

Current date and time are determined from the cellular network or satellite position fix. They cannot be specified manually.

The MI-8 does not contain a real-time clock or backup battery to maintain time when powered off. Data logging is disabled at power up until date and time can be determined.

Date and time from the cellular network have priority and are used when available. Satellite based time and date are used if a cellular connection cannot be established. When cellular network time is used the displayed time and the timestamps within logs reflect the local time zone. All times are UTC when satellite-based time is used.

Triggers

Captured data is organized into 'Triggers'. Each trigger defines the conditions when data should be recorded in a log and/or pushed to a remote server. The trigger also defines which measurement channels are recorded and pushed. Up to three triggers are supported.

Each trigger includes a *trigger[].channels* configuration parameter. This parameter contains a comma separated list of channels which will be recorded into data logs and/or pushed to a remote server. For example, the string: "vbat, tc0" specifies the battery voltage and thermocouple 0 temperature should be captured by the given trigger. See Table 9 – Avaliable Data Measurement Channels for a list of available channels. The order of channels in this list do not matter and white space is ignored.

Each trigger also contains a *trigger[].condition* parameter. This condition is an algebraic expression that determines when the trigger is 'active' based on the state of one or more MI-8 inputs. Data is only logged and only pushed to a remote server when the trigger is active. By creating triggers with custom conditions, the MI-8 is able to guarantee important events are captured while minimizing network bandwidth other times. For example, one trigger with an 'always-on' condition may push all measured channel values to a server once a day and a second trigger may be configured to push data to the server every 10 seconds if a particular thermocouple temperature is high.

Triggers are considered 'active' if the *trigger[].condition* algebraic expression evaluates to a non-zero value. The condition may contain numbers, operators, and MI-8 channel names (see Table 9). Whitespace is ignored. The following table describes the operators (symbols) supported within the condition field.

	Name	Return Value
+	*Addition*	Value of left operand plus the right
-	*Subtraction*	Value of left operand minus the right
*	*Multiplication*	Value of left operand multiplied by the right
/	*Division*	Value of left operand divided by the right
&	*Logical AND*	1 if both left and right operand is true
|	*Logical OR*	1 if either left or right operand is true
<	*Less than*	1 if left operand is less than the right
>	*Greater than*	1 if left operand is greater than the right
=	*Equals*	1 if left operand is equal to the right
( )	*Parenthesis*	Value of the expression between the (and)

Table 6 – Valid Trigger Condition Operators

Table 7 describes example of possible conditions.

Condition	Description
"1"	Always true. Use this condition to log data anytime the MI-8 is powered
"an0 > 2.5"	Trigger active any time the voltage on the an0 input is greater than 2.5 Volts.
"(tc0 > 35.0) | (tc1 > 22.0)"	Trigger is active whenever either the thermocouple 0 (tc0) temperature is greater than 35°C or thermocouple 1 (tc1) is greater than 22°C
(35.919<lat) & (lat<36.378) & (-115.369<lon) & (lon<-114.921)	Trigger is active whenever the MI-8 is in Las Vegas, Nevada

Table 7 – Example Trigger Conditions

Logging

Data from each trigger is saved into separate log files. The logs are saved in plain text as comma separated values (CSV) which can be viewed using common spreadsheet software. The first line (row) of the file identifies the data column names. Each subsequent row describes a single data point.

The first column indicates the data point date in YYYY-MM-DD format (Year – month – day). The second column contains the time of the sample in HH:MM:SS.SSS (Hours : minutes : seconds . milliseconds).

The data sample period (time between samples in the log file) is specified by the *trigger[].log_period* configuration parameter. Logging is disabled if *log_period* is zero. If *log_period* is non-zero, a sample is captured and appended to the log file each sample period (in seconds) as long as the corresponding trigger is active.

An optional pre- and post-trigger delay may be applied to the trigger logging period. The *trigger[].stop_delay* parameter specifies the amount of time in seconds to continue logging after a trigger becomes false. The *trigger[].start_delay* parameter specifies the amount of time to wait after the trigger becomes active before beginning to log data.

If the *trigger[].append_log* parameter is true, the MI-8 will append data from each trigger event to the same file. Otherwise, the MI-8 will create a separate log file for each time a trigger becomes active. In either case, a new file will be created if any log grows too large (exceeds *trigger[].max_log_size* bytes). If max_log_size is not specified, log files are split once they grow larger than 32MB.

Log files are placed in a separate folder (based on the trigger name) for each trigger. Each filename begins with the trigger name (*trigger[].name*). If *trigger[].append_log* is false, the date and time is then appended to the end of the file name. Finally, if any log file grows too large and is split, a four-digit suffix is appended and will increment each subsequent time the log is split. Table 8 Describes possible log file names, depending on MI-8 configuration.

Log Filename	**Description**
{trigger[].name}.csv	*append_log*=true; Data is appended to the same log file each time the trigger becomes active. This is the first log file created.
*{trigger[].name}*_0001*.csv*	*append_log*=true; Data is appended to the same log file each time the trigger becomes active. The log file was split twice because the previous files exceeded *max_log_size.* This is the second log file to be created for this trigger.
*{trigger[].name}*_20220103_040506*.csv*	*append_log*=false; Unique log file is created each time the trigger becomes active. This trigger became active at 4:05:06 on January 3rd, 2022. This is the first log file created for the trigger that became active at this time.
*{trigger[].name}*_20220103_040506_0002*.csv*	*append_log*=false; Unique log file is created each time the trigger becomes active. This trigger became active at 4:05:06 on January 3rd, 2022. The log file was split twice because the previous files exceeded *max_log_size.* This is the third log file to be created for the trigger that became active at this time.

Table 8 – Example Log Filenames

PC Connection

The MI-8 may be configured to act as a USB mass storage device (a USB stick). When the PC is connected, the PC has exclusive access to the SD card and the MI-8 is unable to log data.

Set the *usb_mass_storage* configuration parameter to false to disable this behavior and allow the MI-8 to continue logging data when USB is connected. Note that the PC will no longer be able to read or write to the SD card when *usb_mass_storage* is flase. This mode allows the MI-8 to be powered through USB rather than the dedicated two-pin power connector and still be able to log data to the SD card.

DAQ Channel names

Table 9 describes the available MI-8 measurement channels which may be logged and pushed to a remote server. Note that not all channels are available at the same time. For example, rtd0-3 and res0-3 are not present when six thermocouple inputs are enabled (since they use the same physical MI-8 pins). Channels are only shown on the onboard display if they are enabled through the MI-8 configuration file. The ‘Related Parameter’ column of Table 9 indicates which parameters determine whether each channel is enabled. Channels are always available if the ‘Related Parameter’ column is blank.

Channel Name	**Description**	**Unit**	**Related Parameter**
tc0	Thermocouple channel 0	Degrees Celsius	*num_thermo*
tc1	Thermocouple channel 1	Degrees Celsius	*num_thermo*
tc2	Thermocouple channel 2	Degrees Celsius	*num_thermo*
tc3	Thermocouple channel 3	Degrees Celsius	*num_thermo*
tc4	Thermocouple channel 4	Degrees Celsius	*num_thermo*
tc5	Thermocouple channel 5	Degrees Celsius	*num_thermo*
ambient	Ambient temperature of DAQ	Degrees Celsius	
an0	Auxiliary analog input 0	Volts	
an1	Auxiliary analog input 1	Volts	*num_rtd*
an2	Internal header analog input 2	Volts	*use_aux_header*
an3	Internal header analog input 3	Volts	*use_aux_header*
an4	Internal header analog input 4	Volts	*use_aux_header*
vbat	MI-8 supply voltage	Volts	
rtd0	RTD sensor temperature 0	Degrees Celsius	*num_rtd*
rtd1	RTD sensor temperature 1	Degrees Celsius	*num_rtd*
rtd2	RTD sensor temperature 2	Degrees Celsius	*num_rtd*
rtd3	RTD sensor temperature 3	Degrees Celsius	*num_rtd*
res0	Resistance input 0 (rtd0)	Ohms	*num_rtd*
res1	Resistance input 1 (rtd1)	Ohms	*num_rtd*
res2	Resistance input 2 (rtd2)	Ohms	*num_rtd*
res3	Resistance input 3 (rtd3)	Ohms	*num_rtd*
lat	GPS latitude	Degrees (decimal)	*gnss_period*
lon	GPS longitude	Degrees (decimal)	*gnss_period*
alt	GPS altitude	Meters	*gnss_period*
di0	Internal header digital input 0	none	*use_aux_header*
di1	Internal header digital input 1	none	*use_aux_header*
di2	Internal header digital input 2	none	*use_aux_header*
rssi	Cellular signal strength	dBm	

Table 9 – Avaliable Data Measurement Channels

Electrical Connections

Power Supply to MI-8

The MI-8 may be powered through the micro-USB connector or through the 2-pin Molex miniFit connector.

The 2-pin connector supports a wide supply voltage range. It can be used with fixed 5, 12, or 24VDC power supplies, or it may connect to an external battery. The MI-8 will enter a low-power mode when the supply voltage falls below a configured threshold to avoid over discharging a battery. This low power state is not a replacement for a true battery management system. All rechargeable battery chemistries require a protection circuit to electrically disconnect the battery from all loads once the battery is completely discharged.

The operating environment must be considered when selecting a battery for a battery powered installation. Many battery chemistries, especially lithium-based, cannot tolerate freezing temperatures and are dangerous at hot temperatures. Lead-acid is often the most robust choice in outdoor installations, but care must still be made to ensure the battery does not over discharge and freeze. Contact technicalsupport@fusiondaq.com for more information and for design support.

The MI-8 will enter a low power state if USB power is disconnected and the voltage at the two-pin connector falls below the threshold defined by *the sleep_voltage* configuration parameter. The MI-8 will resume normal operation once the supply voltage rises above wake_voltage (or if USB power is provided). The *sleep_voltage* and *wake_voltage* parameters should differ by at least 0.1 Volts to provide hysteresis and prevent the MI-8 from entering a cycle of waking then shutting back down when the battery voltage nears the sleep threshold.

Set *sleep_voltage* and *wake_voltage* to 0 to disable low power mode.

Regulated Power Output

The MI-8 includes an integrated 14.5V DC regulated power output which may be used to power external sensors or loads.

Regulated power is available, even if the MI-8 is powered by a higher or lower supply voltage. The integrated supply will turn off when the MI-8 powers down. See the *sleep_voltage* and *wake_voltage* configuration parameters.

Custom output voltages are available. Contact sales@fusiondaq.com for more information.

Thermocouple Measurements

The MI-8 inputs are configured in two banks. Each bank is capable of measuring three thermocouples or two RTD/resistances. The MI-8 may be configured to measure zero, three, or six thermocouples at a time. See the *num_thermo* and *num_rtd* parameters in the configuration file. Thermocouple channel 0 (the first channel) it the top left-most pair of connections on DAQ. Thermocouple connections are labeled ‘TC0’ through ‘TC5’ on the MI-8 enclosure.

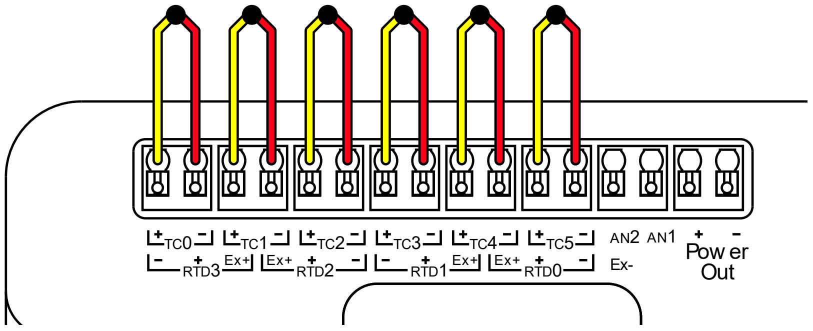


Figure 5 – Thermocouple Connections to MI-8

Thermocouples are polarized. There is a positive and negative wire and each must be connected to the correct positive or negative terminal of the DAQ to properly function. If the polarity of the thermocouple is not known, connect it to the DAQ at random then warm the thermocouple. If the measured thermocouple temperature is above the measured ambient temperature on the DAQ the thermocouple is connected correctly. If the measured temperature is below ambient the thermocouple polarity is reversed.

Thermocouples work by producing a small voltage that is proportional to the temperature difference between the hot-junction (where the two wires meet) and the PUSH-IN connector on the DAQ (the cold-junction). Thermocouples measure a 'delta' or relative temperature, not an absolute. Rapid fluctuations in DAQ temperature will contribute to measurement error. If there is a four-degree temperature difference between the PUSH-IN connectors and the internal DAQ temperature sensor, that offset will result in an additional four degrees of error in the absolute measured thermocouple temperature. For best performance, place the DAQ the out of direct sunlight in a location that is protected from rapid changes in temperature.

The MI-8 is designed to measure differential signals within ±2.4V of ground. In other words, the DAQ can measure grounded thermocouples such as those mounted to a grounded chassis or engine manifold. Thermocouples are measured using a 24-bit, high precision analog to digital converter (ADC).

Thermocouple channels are identified as 'tc0', 'tc1', 'tc2', through 'tc5'. These same names are used in the configuration JSON file, on the DAQ display and in the strings pushed to a remote server. If a thermocouple is disconnected, the display will report the channel as 'OPEN' and a temperature of -999 will be reported to the remote server and log file.

Strain Gauge, RTD, and Resistance Measurements

Strain gauges and RTD temperature sensors may both be thought of as variable resistors. Their electrical resistance changes as they are stretched (strain gauges) or their temperature changes (RTDs). The MI-8 is capable of measuring approximately between 0 and 4000 Ohms from any resistive sensor with 24-bits of resolution. Electrically, each sensor is connected in a similar way.

The MI-8 may be configured to measure zero, two, or four resistive sensors at a time. See the *num_thermo* and *num_rtd* parameters in the configuration file. Resistance channel 0 (the first channel) is the top right-most connection on DAQ. Connections are labeled 'RTD0' through 'RTD3' on the face of the MI-8 enclosure. Note that that resistance measurement input names are reversed from the thermocouples. Resistance inputs are numbered right to left; thermocouples are left to right.

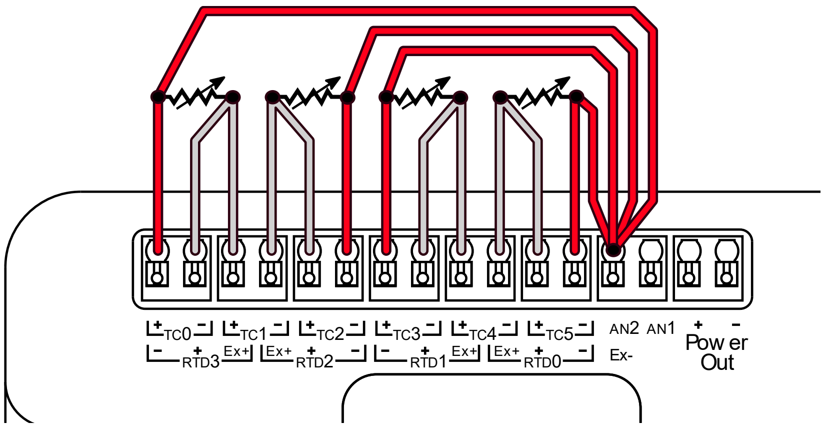


Figure 6 – Resistance Measurement Electrical Connections

Resistance is measured indirectly by passing a small current (less than 2mA) through a sensor then measuring the voltage drop across the sensor. Resistance is then calculated using Ohm's law (R = V/I). If the resistance of the wires or leads between the sensor and the MI-8 is low, and the sensor resistance is relatively high, the lead (wire) resistance may be ignored. If, however, the sensor resistance is relatively low or the lead resistance is high because (the lead wires are long) the lead wire resistance can introduce significant measurement error.

Regardless of the number of wires connected between a resistive sensor and the MI-8, four MI-8 pins are used. Excitation current flows out through the Ex+ pin and into the Ex- pin. Voltage is measured between the + and - pins. Each sensor uses unique Ex+, + and - connections. The Ex- pin is shared for all resistive sensors. The An1 analog input doubles as the Ex- pin for resistive measurements and cannot be used as an analog voltage input when the MI-8 is configured to measure resistive sensors.

The MI-8 supports two, three, and four wire connections to sensors. The connection type is specified for each resistive input using the *rtd_connection* configuration parameter.

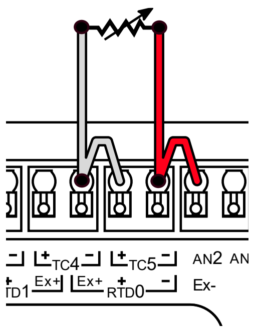


Figure 7 - 2-Wire RTD Connection

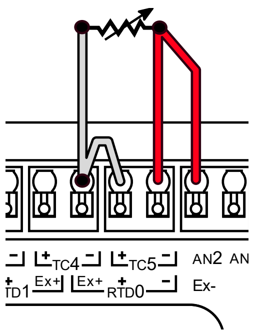


Figure 8 - 3-Wire RTD Connection

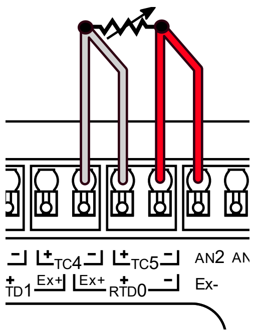


Figure 9 - 4-Wire (Kelvin) RTD Connection

Four-wire sensor connections (Figure 9), also known as Kelvin connections, provide the most accurate resistance measurement. Electrical current only flows through the Ex+ and Ex- wires. Since no significant current flows through the + and – measurement leads, electrical resistance in the sensor wiring will not result in a voltage drop and will not lead to measurement error.

Two-wire sensor connections use the same wire for excitation current and voltage measurement. Since current flows through the wires there will be a voltage drop and that drop will result in a measurement error. If the wire resistance is small or the wire length is short this error may be negligible. When using a two-wire connection both the Ex+ and + and the Ex- and - pins must be connected together as close to the MI-8 connector as possible. See Figure 7 - 2-Wire RTD Connection.

Three-wire connections represent a compromise between four and two-wire connections. Three-wire measurements can be more accurate than two-wire if the resistances of each of the three wires are closely balanced. The MI-8 compensates for lead wire resistance by measuring the resistance between the Ex- and - leads, then subtracting it from the sensor resistance measurement. When using a three-wire connection, the Ex+ and + pins must be connected together as close to the MI-8 input connector as possible. See Figure 8 for an example.

Measured resistance channels are identified as ‘res0’, ‘res1’, ‘res2’, and ‘res3’. These same names are used on the MI-8 display, in log files, and from withing the configuration file.

RTD Measurements

RTD sensors are variable resistors with a calibrated temperature to resistance relationship. The *rtd_type* configuration parameter is used to specify the RTD sensor type and the transfer function to be used to calculate sensor temperature. The MI-8 supports PT100 and PT1000 sensors.

Measured sensor temperatures are identified as ‘rtd0’, ‘rtd1’, ‘rtd2’, and ‘rtd3’ on the MI-8 display, in log files, and from withing the configuration file.

See the Strain Gauge, RTD, and Resistance Measurements section for electrical connection information.

Internal Auxiliary Connector

Three additional analog inputs (‘an2’, ‘an3’, & ‘an4’) and three digital inputs (‘di0’, ‘di1’, & ‘di2’) are available directly on the MI-8 circuit board. Each input may be used to initiate triggers and may be logged and/or pushed to a remote server.

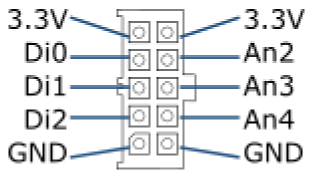


Figure 10 – Auxiliary Connector Pinout

The auxiliary inputs are not normally included on the MI-8 display. They may be enabled by setting the *use_aux_header* configuration parameter to true. The analog inputs are capable of measuring 0 to 3.3V with 10-bit resolution. The digital inputs will report ‘1’ if the pin voltage is near 3.3V and ‘0’ if the voltage is near GND.

Week pull-up or pull-down resistors may be selected for each of the digital inputs independently. The pull up/down configuration is specified using the *dig_in_mode* parameter. If *dig_in_mode* is set to ‘pulldown’ the digital input state will be zero when the input is disconnected. The input will be one when *dig_in_mode* is set to ‘pullup’ and the input is disconnected. Otherwise, if *dig_in_mode* is set to ‘float’, no pull-up or pull-down is connected and the measure pin digital state is undefined when the input is disconnected.

Apply caution when using the auxiliary connector. There is no protection circuitry between the connector and the microprocessor. To prevent permanent damage, do not exceed 0-3.3V and any digital or analog input and do not short the 3.3V output to GND.

Memory Card

A micro-SD card is used to store the MI-8 configuration settings as well as data logs. The card must be present in order for the MI-8 to boot.

Alternative third-party SD cards are supported so long as they support SPI communication and are rated at 3.3V operation.

The memory card included with the MI-8 is rated for used at -40 – 85°C. Most consumer grade SD cards are not rated for this temperature range. Ensure the SD card temperature range is sufficient if the MI-8 is mounted in an outdoor or industrial environment and an alternative card is used.

SIM Card

The MI-8 is tested with Hologram SIM cards. Factory testing and installation of alternative cards is available. Contact sales@fusiondaq.com for more information.

The MI-8 requires 1.8V compatible SIM cards. Similar to SD cards, most consumer grade SIMs do not support the full -40 – 85°C industrial temperature the MI-8 is rated for. Ensure the operating temperature range is sufficient when using an alternative SIM

The MI-8 enclosure must be disassembled in order to remove and replace the SIM. Always disconnect power from the MI-8 before removing the SIM.

Antenna Selection

Separate antennas are required for GNSS (GPS) and cellular connectivity. The MI-8 enclosure provides two female SMA antenna connectors. The MI-8 circuit board has male U.FL connectors.

An active antenna (built in LNA) is required for GNSS operation. The MI-8 supplies 3.8V DC at the GNSS antenna port to power an active antenna.

A passive antenna is preferred for the cellular connection. No DC power is provided at the MI-8 cellular antenna port to supply an active antenna.

Network Communication

Measured data may be pushed to a remote server using HTTP or MQTT network protocols. The transmission period and the data to be sent is configured for each MI-8 data trigger. See section Triggers.

Sever communication settings are specified in the *push* configuration parameters. Some parameters (*mode, server, port*, …) are used by all communication protocols (HTTP or MQTT).

The *push.server* parameter specifies the DNS name or IP address of the server to communicate with. This string should not include a protocol prefix (HTTP://, MQTT://, etc) and should not contain a port number or specific file or path. The connection port number is specified by *push.port*. Common port numbers are described in Table 10.

Protocol	**Secure (*use_ssl*)**	**Port**
HTTP	False	80
HTTPS	True	443
MQTT	False	1883
MQTT	true	8883

Table 10 – Standard Network Communication Ports

SSL encryption is available for HTTP and MQTT connections and is enabled using the *push.use_ssl* configuration parameter. SSL encrypts data sent from the MI-8 to the remote server. Note that SSL communication requires extra cellular network data. Use encryption to protect sensitive data. Disable encryption to minimize cellular data usage.

The MI-8 name (specified by the *name* parameter) is included in the data sent to the remote server when the *push.include_name* configuration parameter is true. This feature may be used to differentiate data from multiple MI-8 devices. This feature may be disabled to reduce cellular data usage.

The MI-8 will track the minimum and maximum value of each measured channel when the push.track_min_max configuration parameter is true. The minimum/maximum envelop is reset each time data is successful pushed to a server. This feature allows the MI-8 to capture peaks and short duration measurement extremes without requiring frequent server notifications. Two extra signals are included in the push notification for each physical channel. The minimum value signal name is the name of the physical measurement channel with a '_min' suffix. The maximum signal has a '_max' suffix. For example, if the 'tc0' channel is included in a trigger and *track_min_max* is true, the server notification will include the 'tc0', 'tc0_min', and the 'tc0_max' signals.

MQTT

MQTT is a lightweight network protocol frequently used in IOT (Internet of Things) applications. When the *push.mode* parameter is 'mqtt' the MI-8 will publish measurement values to a MQTT broker (server).

If the MQTT broker requires authentication, the *push.username* and *push.password* configuration parameters may be used to specify the user credentials. Set these parameters to an empty string (two quotes with no characters in between) if either parameter is not required by the broker.

All measurement channels are published to a single MQTT topic using a JSON formatted string. See JSON File Format for more information. The telemetry topic name is specified by the *push.path* parameter, the device attribute topic name is specified by the *push.attributes_path* parameter.

HTTP GET

Measured values are pushed to a remote server using HTTP GET (encoded into the server URL request) when the *push.mode* configuration parameter is 'get'. Data is encoded as key/value pairs within the URL. The measurement channel names are the keys and the measurement values are the values.

The measurement timestamp is pushed using the "ts" key and is encoded as a UNIX timestamp (epoch) times 1000. The timestamp is the number of milliseconds since January 1st, 1970.

The "name" key is included if the *push.include_name* parameter is true.

HTTP POST

Measured values are pushed to a remote server using HTTP POST when the *push.mode* configuration parameter is 'post'. Data is encoded as HTML form key/value pairs when the *push.use_json* parameter is false, or is encoded in the body of the request as a JSON string if the parameter is true. See JSON File Format for more information.

The measurement timestamp is pushed using the "ts" key when *push.use_json* is false. The timestamp is the number of milliseconds since January 1st, 1970.

The "name" key is included if the *push.include_name* parameter is true.

JSON File Format

JSON is an open data format commonly used in Web applications. Wikipedia provides an excellent overview of the standard and its syntax: https://en.wikipedia.org/wiki/JSON

JSON is used for the MI-8 configuration file and may also be used to encode the telemetry data sent to a remote server.

JSON can be thought of a collection of key/value pairs. Keys are always strings. They are followed by a colon, and finally a value which can be a string, numeric value, array of values, or a sub collection of more key/value pairs (an object).

For example, "name":"MI-8" assigns the string value "MI-8" to the key "name".

Key/value pairs are separated by commas.

String values are surrounded by double quotes ("). An empty string is represented by two quotes with no spaces or characters between them.

Arrays are surrounded by square brackets [], and contain a list of comma separate numbers, strings, or objects.

Objects are surrounded by braces {} and contain a comma separated list of more key/value pairs

Datatype	Examples	Description
String	“My String” “MI-8”	Letters, numbers, and some special characters, all surrounded by double quotes.
Number	123 12.34 -0.8 .999	Number, with or without a decimal point
Array	[“apple”, “banana”, “orange”] [1, 2, 3.5, 8] [42]	Collection of strings, numbers or objects, separated by commas and surrounded by square brackets []
Object	{“name”:”MI-8”, “count”:10}	Collection of key/value pairs separated by commas, and surrounded by curly braces {}. This object contains a string called “name” and a number called “count”

Figure 11 - Example JSON datatypes

JSON Telemetry Format

Telemetry refers to measured data which changes over time. In general, telemetry data is transmitted each time data is sampled while at least one trigger is active. Data is encoded using JSON when the *push.use_json* parameter is true.

```
{
  "ts":000000000000,
  "name":"ExampleName",
  "imei":"860000000000000",
  "iccid":"8900000000000000000",
  "values":{
    "tc0":100.0,
    "tc0_min":100.0,
    "tc0_max":100.0
  }
}
```

Figure 12 - Example JSON Telemetry Data

JSON formatted telemetry always contains at least two top-level keys: “ts” and “values”. The “ts” key is the timestamp when the data was sampled. This is the time the data was measured, not necessarily the time it was successfully received by the server. Timestamps are the number of milliseconds since January 1st, 1970.

The ‘values’ key is an object (list of more keys) which contains measured values from each channel from each active trigger. Channels are only included if they are part of an active trigger and the data was measured at least once since the MI-8 powered up. For example, GNSS position may take several minutes to determine and won’t be included in telemetry pushes until the MI-8 has a valid position fix. In cases where a trigger became active, but not all measurement channels were ready, the delayed channel values will be transmitted as soon as they become available, even if their respective trigger is no longer valid.

JSON telemetry data may also contain the “name”, “imei”, and “iccid” keys, depending on *push* configuration parameters. The “name” value is specified by the “name” parameter in the “config.json” configuration file. The “imei” and “iccid” keys provide the cellular modem IMEI and the SIM ICCID numbers respectively.

The “name”, “imei”, and “iccid” keys may be included at the expense of additional cellular data usage in applications where separate attributes notification cannot be processed. When possible, the name, IMEI, and ICCID should only be transmitted in attribute notifications since attributes are transmitted once per power cycle while telemetry data is transmitted each sample.

JSON Attributes Format

Device attributes refer to data which doesn’t change over time. Aattributes are only pushed to a remote server once, when the MI-8 first powers up an connects to a cellular network. Data is encoded using JSON when the *push.use_json* parameter is true.

```
{
  "name":"ExampleName",
  "imei":"860000000000000",
  "iccid":"8900000000000000000",
  "firmware": "2023.01.07"
}
```

Figure 13 - Example JSON Attributes Data

The “name”, “imei”, and “iccid” keys within the device attributes contain the same values of the respective keys in the telemetry data (if included). The “name” value is specified by the “name” parameter in the “config.json” configuration file. The “imei” and “iccid” keys provide the cellular modem IMEI and the SIM ICCID numbers.

The “firmware” key describes the MI-8 firmware revision and generally contains the date the firmware was released.

Configuration File

The MI-8 is configured using a JSON formatted text file in the root directory of the SD card named "config.json".

The SD card must be formatted using a FAT32 filesystem.

This filename is case sensitive. It can be difficult to change the case of filenames on Fat32 filesystems within Windows. If the case of any character in the filename is wrong, change the filename to something different such as "temp.json" then change it back to "config.json" with all lowercase letters.

JSON Configuration File Parameters

Parameter	**Description**
name	*string* (32 characters max) Descriptive name used to identify a particular DAQ. The name may be included when pushing data to a server to identify the source of the data. See push.include_name parameter. Valid characters include letters, numbers, underscore (_) and hyphen (-)
sleep_voltage	*number* Minimum operating voltage. DAQ will enter a low power state and stop logging and pushing data to server if the DAQ is not powered through USB and the supply voltage (2-pin Mini-Fit connector) falls below this threshold. Valid range: 4.0 - 30.0 Volts
wake_voltage	*number* The DAQ will powerup if the supply voltage is above wake_voltage or if power is supplied through the USB port. Wake_voltage should be at least 0.1 volt greater than sleep_voltage to prevent the DAQ from rapidly cycling on and off when the supply voltage is near the two thresholds. Valid range: 4.0 - 30.0 Volts
display_sleep	*number* Number of seconds DAQ display remains powered (active) after the last key press. The display will turn off (DAQ will remain powered) if no buttons are pressed for this number of seconds. The display will remain powered at all times if display_sleep is 0. Parameter is used to reduce power consumption in batter powered installations.

	Valid range: 0 – 65535 Seconds
gnss_period	*number* GNSS (GPS) sample period in seconds. GNSS measurements are disabled if this parameter is zero. Note that the cellular data connection is interrupted while the GPS modem is active. Valid range: 0 – 65535 Seconds
usb_mass_storage	*boolean* The DAQ will appear as an USB drive when connected to a PC through USB if this parameter is true. If this parameter is false, the DAQ will still draw power through the USB port, but will not appear as an external drive. The SD card cannot be accessed by the DAQ and a PC at the same time. Setting this parameter to false disables the PC connection and allows the DAQ to continue accessing the SD card while powered through the USB port. Valid values are: • "true": The DAQ will appear as a USB drive when connected to a PC • "false": USB port used for power only
num_thermo	*number* Number of thermocouple sensors connected to the DAQ. Valid values are: 0, 3, or 6
thermo_type[]	*array of enumerations (strings)* Type (J, K, T, etc) of each thermocouple connected to the DAQ. This value determines which look-up table is used to convert thermocouple voltage to temperature. The first element in the array corresponds with TC0. The second element corresponds with TC1, etc. Array should contain num_thermo elements. For example, if three thermocouples are used: thermo_type:["j","k","k"] Valid values are: "b", "e", "j", "k", "n", "r", "s", or "t": (default)
num_rtd	*number* Number of RTD or resistive sensors connected to the DAQ, including strain gauge measurements. Valid values are: 0, 2, or 4

rtd_type[]	*array of enumerations (strings)* Type (PT100, PT1000, etc) of RTD sensor connected to the DAQ. This value determines which look-up table is used to convert measured resistance to temperature. The first element in the array corresponds with TC0. The second element corresponds with TC1, etc. Array should contain num_rtd elements. For example, if two RTD sensors are used: rtd_type:[“pt100”,”pt1000”] Valid values are: • “pt100”: (default) • “pt1000”:
rtd_connection	*array of numbers* Number of wires used to connect the DAQ to each RTD or resistive sensor. Sensors are typically connected with two, three or four wires. Array should contain num_rtd elements. For example, if two RTD sensors are used: rtd_connection:[3,4] Valid values are: 2, 3, or 4. Default is 4.
rtd_resistance	number Resistance of the RTD reference resistor on the DAQ circuit board. This resistor is installed during manufacturing and is generally not adjustable by the end-user. Valid values are from 0 to 100,000. Default value is 4000
dig_in_mode[]	*array of enumerations (strings)* Pull up/down configuration for each auxiliary digital input. Each pin may be internally pulled to GND, 3.3V, or allowed to float (no pull up or pull-down resister). The first element in the array should corresponds with di0. The second element corresponds with di1, etc. Array should contain three elements. Valid values are: • “pullup”: Digital input weakly pulled high to 3.3V • “pulldown”: Digital input weakly pulled to GND • “float”: (default) No Pullup or pull-down resister enabled
use_aux_header	*boolean*

	Specifies whether or not analog inputs from internal auxiliary pin header are present on display and available for logging. Valid values are: • "true": Internal analog and digital inputs included on display • "false": Internal inputs hidden on display
flip_display	*boolean* Specifies whether or not the onboard display should be flipped (upside down). Valid values are: • "true": Render display upside down – PUSH-IN connectors on bottom • "false": (default) PUSH-IN connectors above display
network	*object* JSON object used to consolidate cellular network connection settings
network.apn	*string (32 characters max)* Name of the cellular network access point (APN). This name is generally defined by the SIM card wireless carrier.
network.username	*string (32 characters max)* Optional username used to connect to the wireless network. This string is specified by the wireless carrier. Specify an empty string (two quotation marks with no characters in between) if no username is required.
network.password	*string (32 characters max)* Optional password used to connect to the wireless network. This string is specified by the wireless carrier. Specify an empty string (two quotation marks with no characters in between) if no username is required.
network.mode	*enumeration (string)* LTE connection type (CAT-M, IOT-NB, or both) Valid values are: • "cat-m": Connect to cellular network using CAT-M protocol • "iot-nb": Connect to cellular network using IOT-NB protocol • "both": (default) Connect using IOT-NB or CAT-M protocol
network.bands_cat_m	*Array of numbers* LTE bands to be used when connecting to the cellular network via CAT-M. More bands may allow the MI-8 to reach more cell towers, but will require more time to connect as the MI-8 must search more bands for the best connection. See Table 4 – Cellular Modem Specifications for a list of supported CAT-M bands.

	Default Value: 2, 4, 12, 13, 66
network.bands_cat_nb	*Array of numbers* LTE bands to be used when connecting to the cellular network via IOT-NB. More bands may allow the MI-8 to reach more cell towers, but will require more time to connect as the MI-8 must search more bands for the best connection. See Table 4 – Cellular Modem Specifications for a list of supported IOT-NB bands. Default Value: 2, 4, 12, 13, 66
push	*object* JSON object used to group remote server connection settings. The remote server receives measured data from the DAQ
push.mode	*enumeration (string)* Protocol used to transmit (push) data from DAQ to remote server. Specify "none" if data should not be pushed to server. Valid values are: • "mqtt": • "post": • "get": • "none": (default)
push.path	*string (64 characters max)* MQTT topic to publish telemetry values if push.mode is "mqtt". HTTP path to send telemetry values if push.mode is "get" or "post". The HTTP path is the string after the server name in a URL and should not include the leading slash ('/').
push.attributes_path	*string (64 characters max)* MQTT topic to publish device attributes if push.mode is "mqtt". HTTP path to send device attributes if push.mode is "get" or "post". The HTTP path is the string after the server name in a URL and should not include the leading slash ('/').
push.push_attributes	*boolean* Specifies whether or not device attributes are pushed to a server each power cycle. Attributes are values which generally do not change over time such as the MI-8 name, IMEI, and ICCID. Set value to "false" to reduce data usage. Valid values are: • "true": Push device attributes each time MI-8 powers up • "false": (default) Do not push device atributes

push.server	*string (64 characters max)* MQTT or HTTP server name. The server name is the substring within a URL between the double slashes (//) and the first single slash (/)
push.port	*number* MQTT or HTTP server port number. This number is generally 80 for HTTP connections, 443 for HTTPS (secure), and 1883 for MQTT. Valid values are from 0 to 65535. Default value is 0
push.username	*string (32 characters max)* (MQTT only) Optional username used to connect to MQTT server. Specify an empty string (two quotes with no characters in between) if no username is required.
push.password	*string (32 characters max)* (MQTT only) Optional password used to connect to MQTT server. Specify an empty string (two quotes with no characters in between) if no password is required.
push.use_ssl	*boolean* Specifies whether or not SSL is used to encrypt data transmitted to the MQTT or HTTPS server. This value must be true when connecting to HTTPS servers and false when connecting to HTTP servers. Valid values are: • "true": SSL is used (secure communication) • "false": (default) SSL is not used
push.use_headers	*boolean* Specifies whether the "Cache-control", "Connection", and "Accept" HTTP headers are transmitted as part of GET/POST requests. These headers can generally be excluded to reduce data usage; however they are sometimes needed by some web hosts. The "Content-Type" header is always transmitted, regardless of the *push.use_headers* value. Valid values are: • "true": "Content-Type", "Cache-control", "Connection", and "Accept" HTTP headers included in each HTTP GET/POST request • "false": (default) Only "Content-Type" header transmitted in HTTP requests
push.use_json	*boolean* Specifies DAQ data encoding when pushing data to a HTTP server. Telemetry and attribute data is always encoded using JSON when using MQTT.

	Valid values are: • "true": DAQ values are encoded into a JSON object • "false": (default) DAQ values are specified in separate HTTP name/value pairs.
push.include_name	*boolean* Specifies whether the DAQ name string (specified in DAQ configuration file) is included in data pushed to the MQTT/HTTP server. The string may be excluded to reduce data usage. The name is still transmitted in the attributes data if *push.push_attributes* is "true", even if *push.include_name* is "false" Valid values are: • "true": DAQ name string included in push payload • "false": (default) DAQ name not included
push.include_imei	*boolean* Specifies whether the cellular modem IMEI is included in telemetry data pushed to the MQTT/HTTP server. The string may be excluded to reduce data usage. The IMEI is still transmitted in the attributes data if *push.push_attributes* is "true", even if *push.include_imei* is "false" Valid values are: • "true": Cellular modem IMEI included in telemetry data • "false": (default) IMEI not included
push.include_iccid	*boolean* Specifies whether the SIM card ICCID is included in telemetry data pushed to the MQTT/HTTP server. The string may be excluded to reduce data usage. The ICCID is still transmitted in the attributes data if *push.push_attributes* is "true", even if *push.include_iccid* is "false" Valid values are: • "true": SIM card ICCID included in telemetry data • "false": (default) ICCID not included
push.track_min_max	*boolean* When true, the DAQ will track the minimum and maximum value for each channel since the last push to the server. Valid values are: • "true": Current, minimum and maximum values for each DAQ channel are included in telemetry push payload • "false": (default) Only in current channel values are included in push payload
trigger[]	*array of objects*

	Array of trigger objects. Each trigger specifies a group of DAQ channels, when they are sampled, and how they are to be logged and pushed (transmitted) to a remote server. Array may contain up to three objects
trigger[].name	*string (32 characters max)* Base (prefix) used for log file names for the given trigger. Name of file and directory in which log files for the given trigger will be placed. Valid characters include letters, numbers, underscore (_) and hyphen (-). Do not include slashes.
Trigger[].append_log	*boolean* Determines whether a separate log file is created each time the trigger becomes active or if all data is appended to a single file. Valid values are: • "true": All data is written to a single file • "false": (default) A new log file is created each time the trigger becomes active
Trigger[].max_log_size	*number* Maximum log file size. Log files are split (original file closed and a new one created) each time the log size exceeds this threshold. Valid values are from 1 to 1,073,741,823 bytes (1GB). Default value is 33,554,432 bytes (32MB)
trigger[].condition	*string (256 characters max)* Logical expression used to determine when the trigger is active. It is made up of DAQ channel names, numeric values, and algebraic operators. Whitespace is ignored. The trigger is active if the expression evaluates to a non-zero value. Set the condition to "1" to permanently enable a particular trigger. See Table 6 – Valid Trigger Condition Operators for a list of valid operators. For example, the expression "(an0 > 3.2) & (an0 < 4.0)" will cause the trigger to become active anytime the an0 voltage is between 3.2 and 4.0 volts.
trigger[].channels	*string (256 characters max)* Comma delimited list of channels to be logged to a file or pushed to a server. Whitespace is optional. See Table 9 for a list of valid channel names.

	For example, the expression "tc0, vbat, rssi" will cause measured thermocouple 0 temperature, DAQ supply voltage, and cellular signal strength to be logged and pushed to the remote server.
trigger[].push_period	*number* Describes sample period in seconds (1 second minimum) in which data is pushed to the remote server. Set value to 0 to network push behavior. Valid values are from 0 to 4294967.296 seconds. Default value is 0
trigger[].log_period	*number* Log file sample period in seconds (0.01 second minimum). Set value to 0 to disable logging to the SD card. Valid values are from 0 to 4294967.296 seconds. Default value is 0
trigger[].start_delay	*number* Number of seconds after the trigger becomes active to begin logging data. Positive numbers indicate the log will begin after the trigger event. Negative values are not supported. Valid values are from 0 to 2147483.648 seconds. Default value is 0
trigger[].stop_delay	*number* Number of seconds to continue logging data after the trigger becomes inactive. Set value to 0 to stop logging immediately after trigger becomes inactive. Valid values are from 0 to 4294967.296 seconds. Default value is 0

Table 11 – Configuration Parameters

Table of Figures

Appendix A – Example config.JSON: Six thermocouples and HTTP POST

```
{
   "name":"Example DAQ",
   "thermo_type":["k","k","k","k","k","k"],
   "num_thermo":6,
   "num_rtd":0,
   "dig_in_mode":["pulldown","pulldown","pulldown"],
   "use_aux_header":false,
   "usb_mass_storage":true,
   "sleep_voltage":9.9,
   "wake_voltage":10.5,
   "display_sleep":10,
   "gps_period":0,
   "network": {
      "apn":"hologram",
      "username":"",
      "password":"",
      "mode":"both",
      "bands_cat_m":[2,4,12,13,66],
      "bands_iot_nb":[2,4,12,13,66]
   },
   "push": {
      "mode":"post",
      "path":"api/v1/abcd1234567890/telemetry",
      "attributes_path":"api/v1/abcd1234567890/attributes",
      "push_attributes":true,
      "server":"thingsboard.fusiondaq.com",
      "port":443,
      "username":"",
      "password":"",
      "use_headers":false,
      "use_ssl":true,
      "use_json":true,
      "include_name":false,
      "include_imei":false,
      "include_iccid":false,
      "track_min_max":false
   },
   "trigger":[ {
      "name":"Default",
      "channels":"tc0,tc1,tc2,tc3,tc4,tc5,vbat,rssi",
      "log_period":60,
      "push_period":300,
      "start_delay":0,
      "stop_delay":0,
      "condition":"1",
      "append_log":true
   },
   {
      "name":"ActiveFault",
      "channels":"tc0",
      "log_period":0,
      "push_period":60,
      "start_delay":0,
      "stop_delay":0,
      "condition":"tc0>100",
      "append_log":true
   }  ]
}
```

Figure 14 - Six thermocouple with HTTP POST example config.json

Appendix B – Example config.JSON: Four RTD sensors and MQTT

```
{
   "name":"Example DAQ",
   "num_thermo":0,
   "num_rtd":4,
   "rtd_type":["pt100","pt100","pt100","pt100"],
   "rtd_connection":[4,4,4,4],
   "ref_resistance":4000,
   "dig_in_mode":["pulldown","pulldown","pulldown"],
   "use_aux_header":false,
   "usb_mass_storage":true,
   "sleep_voltage":9.9,
   "wake_voltage":10.5,
   "display_sleep":10,
   "gps_period":300,
   "network": {
      "apn":"hologram",
      "username":"",
      "password":"",
      "mode":"both",
      "bands_cat_m":[2,4,12,13,66],
      "bands_iot_nb":[2,4,12,13,66]
   },
   "push": {
      "mode":"mqtt",
      "path":"v1/devices/abcd1234567890/telemetry",
      "attributes_path":"v1/devices/abcd1234567890/attributes",
      "push_attributes":true,
      "server":"thingsboard.fusiondaq.com",
      "port":8883,
      "username":"",
      "password":"",
      "use_ssl":true,
      "use_json":true,
      "include_name":false,
      "include_imei":false,
      "include_iccid":false,
      "track_min_max":false
   },
   "trigger":[ {
      "name":"Default",
      "channels":"rtd0,rtd1,rtd2,rtd3,lat,lon,vbat,rssi",
      "log_period":60,
      "push_period":300,
      "start_delay":0,
      "stop_delay":0,
      "condition":"1",
      "append_log":true
   },
   {
      "name":"ActiveFault",
      "channels":"rtd0",
      "log_period":0,
      "push_period":60,
      "start_delay":0,
      "stop_delay":0,
      "condition":"rtd0>100",
      "append_log":true
   }   ]
}
```

Figure 15 – Four RTD with MQTT example config.json