USB 8-Channel Sensor Interface Technical Manual

Model 2218 | Rev.1.0.1 | March 2024

SENSORAY | embedded electronics

Designed and manufactured in the U.S.A.

SENSORAY | p. 503.684.8005 | email: info@SENSORAY.com | www.SENSORAY.com

7313 SW Tech Center Drive | Portland, OR 97203

Table of Contents

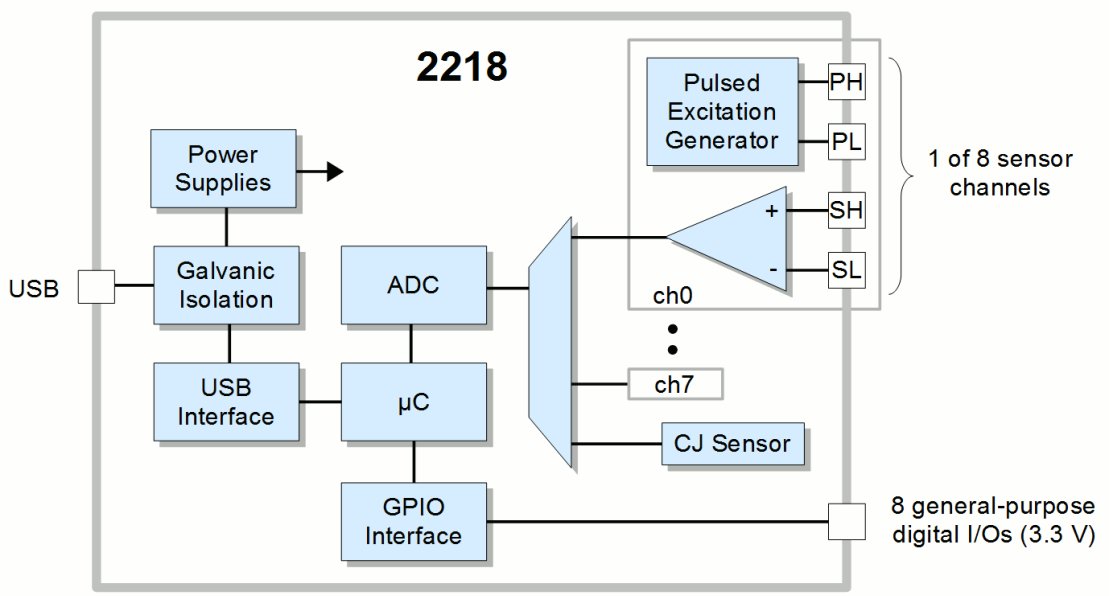
Chapter 1: Introduction....1
1.1 Functional overview....1
1.1.1 Sensor interfaces....1
1.1.2 Digital I/Os....2
Chapter 2: Installation and Operation....3
2.1 Installation overview....3
2.2 Back panel....4
2.2.1 Device address switch....4
2.2.2 Status indicators....4
2.3 GPIO connections....5
2.4 Isothermal bay....6
2.5 Analog connections....6
2.5.1 Best practices....7
2.5.2 RTDs, thermistors and resistors....7
2.5.2.1 Two-wire circuit....8
2.5.2.2 Four-wire circuit....8
2.5.3 Thermocouples....9
2.5.4 Voltage....9
2.6 Measurement noise....9
2.6.1 Warm-up noise....9
2.6.2 Other noise....9
Chapter 3: API....10
3.1 Overview....10
3.1.1 Error codes....10
3.2 Admin functions....10
3.2.1 s2218_OpenApi....10
3.2.2 s2218_CloseApi....11
3.2.3 s2218_OpenDevice....11
3.2.4 s2218_CloseDevice....12
3.2.5 s2218_ErrString....12
3.2.6 s2218_GetApiVersion....13
3.2.7 s2218_GetDeviceVersions....13
3.3 Analog configuration....14
3.3.1 s2218_SetSensorType....14
3.3.2 s2218_SetOpenSensorValues....15
3.3.3 s2218_GetAnalogConfig....16
3.4 Data acquisition....16
3.4.1 Data reports....16
3.4.2 Stream control....17
3.4.3 Functions....17
3.4.3.1 s2218_SetTrigConfig....17
3.4.3.2 s2218_GetTrigConfig....17
3.4.3.3 s2218_SetTrigEnable....18
3.4.3.4 s2218_GetTrigEnable....18
3.4.3.5 s2218_WaitForDataReport....18
3.4.3.6 s2218_ReadDeviceTemp....20
3.5 Alarms....20
3.5.1 s2218_SetLoThreshold....21
3.5.2 s2218_SetHiThreshold....21
3.5.3 s2218_SetLoLimitEnable....22
3.5.4 s2218_SetHiLimitEnable....22
3.5.5 s2218_SetLoAlarmEnable....22
3.5.6 s2218_SetHiAlarmEnable....23
3.5.7 s2218_WaitForLimitAlarm....23
3.6 GPIO....24
3.6.1 Data structures....24
3.6.1.1 Flag byte....24
3.6.1.2 S2218_GPIO_PAIR....24
3.6.1.3 S2218_GPIO_CONFIG....25
3.6.1.4 S2218_GPIO_SETTINGS....25
3.6.2 Functions....26
3.6.2.1 s2218_SetGpioOutputs....26
3.6.2.2 s2218_GetGpioInputs....26
3.6.2.3 s2218_SetGpioConfig....27
3.6.2.4 s2218_GetGpioSettings....27
3.7 Edge capture....28
3.7.1.1 s2218_GpioCapEnable....28
3.7.1.2 s2218_WaitForGpioEvent....29
Chapter 4: Specifications....31
4.1.1 General specifications....31
4.1.2 Sensor specifications....32
Chapter 5: Limited warranty....33

Chapter 1: Introduction

1.1 Functional overview

Model 2218 is a USB-compatible module that interfaces eight sensors to a host computer. It provides excitation for passive sensors and complete signal conditioning for thermocouples, RTDs and thermistors. Each channel can be independently configured to measure voltage, resistance, or any supported sensor type. In addition, eight independent digital I/O lines (3.3 Volt logic) are available for general-purpose use. The module is USB-powered; no external power supply is required.

Figure 1: Model 2218 block diagram



1.1.1 Sensor interfaces

Each sensor channel provides complete signal conditioning for a variety of sensor types, thus allowing any channel to be directly connected to a thermocouple, RTD, thermistor, resistor or DC voltage.

Pulsed excitation is provided for thermistors, resistors and RTDs, which minimizes sensor self-heating and reduces power consumption. Excitation signals are routed to dedicated connector pins to allow users to implement four-wire sensor circuits that eliminate lead-loss errors.

Cold junction correction is automatically applied to thermocouples, and fully differential inputs provide common-mode voltage rejection. Open-circuit detection is automatically activated when a sensor channel is configured to operate with a thermocouple. This is useful for triggering alarms when open sensors are detected, and for forcing the desired system response to fault conditions in closed-loop control applications.

The module automatically scans all sensor channels. As each channel is scanned, the sensor is excited, digitized, normalized to high-precision internal standards, linearized, converted to engineering units, and checked against user-defined alarm thresholds.

The data from select scans are packaged into time-stamped data reports, which are sent to the host computer at a user-defined, periodic rate. The host is automatically notified (via API blocking functions) when data reports are available and when sensor alarm thresholds are exceeded, thereby eliminating the need for host polling.

1.1.2 Digital I/Os

The module's eight general-purpose digital I/Os (GPIOs) may be independently configured to operate as an input or output. The GPIOs use 3.3 V logic levels.

When a GPIO is configured as an input, weak pull-up and pull-down resistors are available which may be optionally enabled. In many cases, these can be used in lieu of external resistors when GPIOs are connected to passive devices such as mechanical switches and optocouplers.

Each GPIO has an independent debounce filter with a filter time of 0 to 255 milliseconds, programmable in 1 ms steps. These filters allow a GPIO to be configured to reject contact bounce and other spurious signals.

Input edge detection is supported on each GPIO. The GPIO pins are automatically sampled at 1 kS/s, which allows pulses as short as 1 millisecond to be detected. The module will automatically notify the host computer (via API function `s2218_WaitForGpioEvent`) when edges have been detected, thereby making it possible to implement an event-driven system and eliminate polling.

Chapter 2: Installation and Operation

2.1 Installation overview

To install the module:

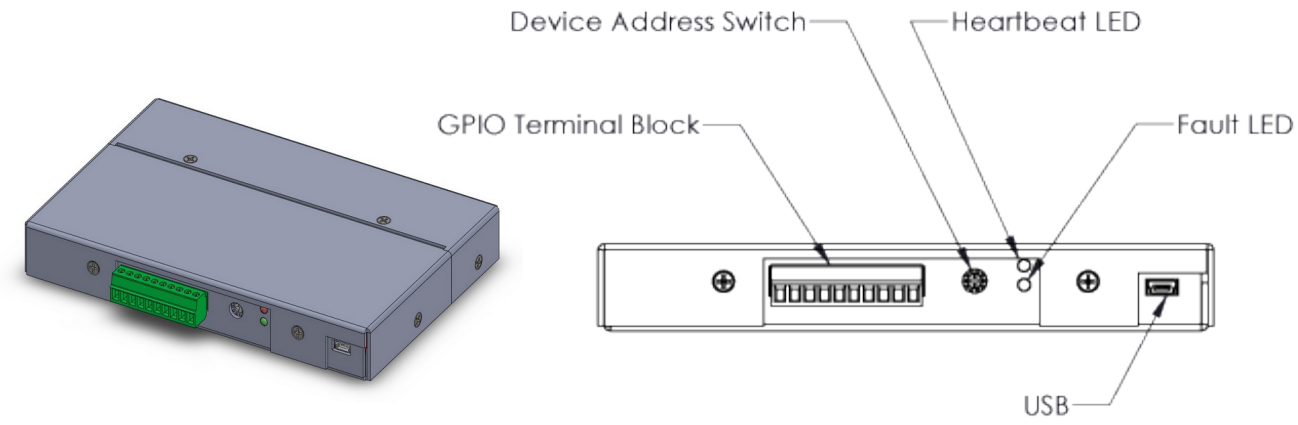
- Set the Device address switch.
- Remove the Isothermal bay cover.
- Connect analog signal wires to the sensor terminal blocks as explained in Analog connections.
- Install the Isothermal bay cover.
- Connect digital signal wires to the GPIO terminal blocks.
- Attach the USB cable.
- Observe the Status indicators to confirm that the module is operating normally.

To remove the module:

- Disconnect the USB cable.
- Disconnect digital signal wires from the GPIO terminal blocks.
- Remove the Isothermal bay cover.
- Disconnect analog signal wires from the sensor terminal blocks.

2.2 Back panel

Figure 2: Model 2218 back panel



2.2.1 Device address switch

A computer may be connected to up to sixteen model 2218 modules directly or through powered USB hubs, or via a combination of these. To facilitate this, each module is assigned an address between zero (factory default) and 15 by rotating its rotary device address switch (see Figure 2) with a small flat-tip screwdriver. A module must be disconnected from USB while its address is being changed.

When calling API functions, the computer communicates with a particular module by specifying the module's address. If a computer will connect to multiple modules then each module must be assigned a unique address. This is done by setting the address switches so that every module is assigned a different address.

2.2.2 Status indicators

Two status LEDs are located on the back panel (see Figure 2): *Heartbeat* and *Fault*. Both LEDs will light for approximately 0.5 seconds when the module is powering up (i.e., when USB is connected) to allow users to verify they are functional. After this, the LEDs are used to indicate various module conditions:

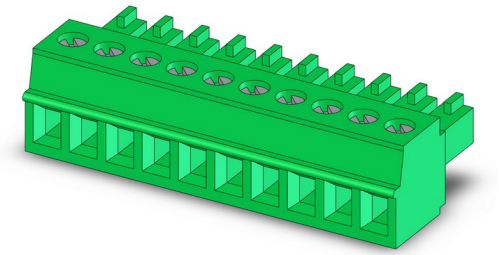
LED		Module condition
Fault	**Heartbeat**	
Off	Flashing	Normal operation
On	Flashing	Analog subsystem is booting or has a persistent fault
Flashing	On	Digital power supply fault
Flashing	Flashing	Analog power supply fault
Flashing	Off	Internal communication error
On	On	Module boot failure
Off	Off	USB disconnected or USB power supply malfunction

2.3 GPIO connections

GPIO signals are accessible at the removable 10-position terminal block located on the module's front panel. This terminal block provides a GND terminal, a +3.3V terminal and one terminal for each GPIO.

GPIO voltages are referenced to the GND terminal, which is isolated from USB GND with a 2 MΩ resistor.

Figure 3: GPIO terminal block



GPIOs are not 5V tolerant and can be damaged if exposed to negative voltages or voltages greater than +3.3V.

Figure 4: TB pinout

GPIO terminal block									
VCC33	GPIO0	GPIO1	GPIO2	GPIO3	GPIO4	GPIO5	GPIO6	GPIO7	GND

Upon power up or module reset, all GPIO pins are configured as inputs by default, with a weak pull-down resistor enabled. The resistor bias can be disabled or changed to pull-up via the API `s2218_SetGpioConfig` function.

Note: it is recommended to use a strong, external pull-up or pull-down resistor if the GPIO load is connected via a long cable or if external noise is likely to be coupled onto a GPIO circuit.

Figure 5: Active-high output

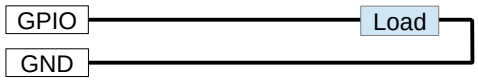
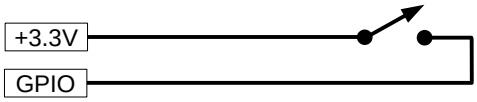


Figure 6: Active-high input w/internal pull-down enabled



2.4 Isothermal bay

Four removable terminal blocks (TBs) are provided for sensor connections. These terminal blocks (TB6-TB9) are located in a recessed isothermal bay near the front of the module.

The isothermal bay is enclosed by a removable cover. When the cover is installed (Figure 7), a narrow slot is exposed between the cover and module body through which sensor wires may pass. This provides a protected, isothermal environment for TB connections, which is essential for high-accuracy thermocouple measurement and for preventing thermocouple effects in other sensor types.

To gain access to the TBs, remove the four hold-down screws and lift the isothermal bay cover as shown in Figure 8. If desired, the TBs may be detached from the module as needed to expedite wiring changes.

After connecting your field wires to the TBs, re-install the isothermal bay cover and secure it in place with four screws.

When securing the isothermal bay cover, do not allow sensor wires to flex the cover as this may damage the cover.

Figure 7: Isothermal bay with cover installed

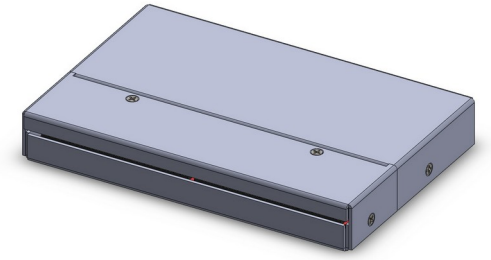
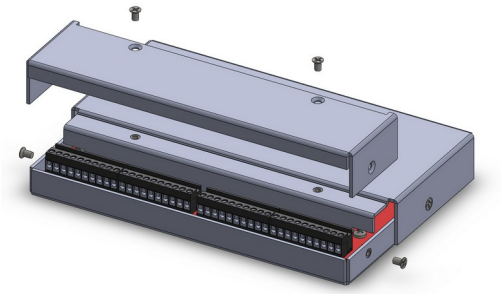


Figure 8: Bay cover removal/installation



2.5 Analog connections

Figure 9: Analog terminal block pinouts (as viewed from front of module)

TB6									
GND	PL0	PH0	PL1	PH1	GND	PL2	PH2	PH3	PL3

TB7									
GND	SL0	SH0	SL1	SH1	GND	SL2	SH2	SL3	SH3

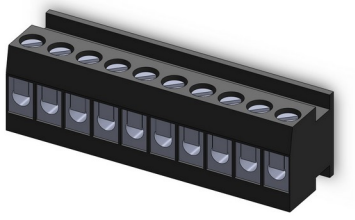
TB8									
GND	SL4	SH4	SL5	SH5	GND	SL6	SH6	SL7	SH7

TB9									
GND	PL4	PH4	PL5	PH5	GND	PL6	PH6	PL7	PH7

Each sensor channel has five screw terminals which are distributed over two terminal blocks:

Signal	Function
SH	Positive voltage sense input
SL	Negative voltage sense input
PH	Positive excitation output
PL	Negative excitation output
GND	Cable shield or signal GND

Figure 10: Analog terminal block (1 of 4)

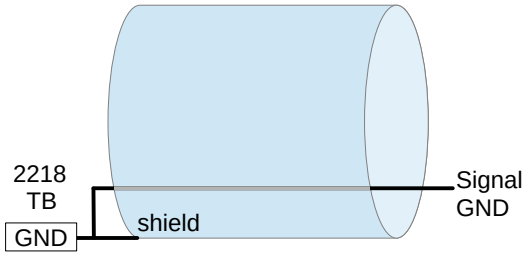


The SH and SL terminals are are used for all sensor types; these are the differential voltage sense inputs. Every voltage source is treated as a differential signal pair. If you are connecting a single-ended source, connect the signal to SH and connect SL to the signal's 0 V ground reference.

Passive sensors also require connections to the PH and PL terminals, which supply positive and negative differential excitation to the sensor. These terminals should be connected only if the sensor requires excitation.The GND terminal may be connected to a cable shield or used as a signal reference voltage, or both.

When GND is connected to a cable shield, the other end of the shield should be left unconnected to avoid ground loops. If both a shield and a signal GND are needed then it is recommended to connect the GND terminal to the shield and to a separate, dedicated signal GND conductor as shown in Figure 11.

Figure 11: Using GND as both shield and signal reference



2.5.1 Best practices

The following practices are recommended for analog field wiring:

- Use shielded cable.
- Connect the GND terminal to the cable shield. Do not connect the other end of the cable shield.
- Use a twisted wire pair for SH and SL.
- Use a twisted wire pair for PH and PL.
- Do not route sensor cables near high-voltage or high-current conductors.
- Avoid long cable runs.

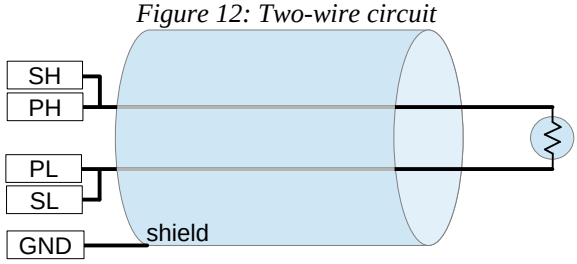
2.5.2 RTDs, thermistors and resistors

RTDs, thermistors and other passive resistive devices may be connected in a two-wire or four-wire circuit.

2.5.2.1 Two-wire circuit

In a two-wire circuit, the SH and PH terminals are shorted together at the module end of the cable, as are SL and PL. This simplifies wiring but introduces error because it forces excitation current to flow in the sense wires, resulting in lead losses (voltage drops across the sense wires). These errors may be significant if high measurement accuracy is required.

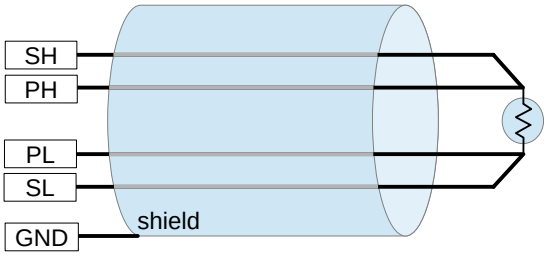
Figure 12: Two-wire circuit



2.5.2.2 Four-wire circuit

In a four-wire circuit, independent excitation and sense wires are routed all the way to the sensor, thus avoiding current in the sense wires and preventing lead loss errors. This is especially important for RTDs, which have relatively low resistance. Thermistors have higher resistance than RTDs over much of their operating range. Consequently, a two-wire circuit may be satisfactory if a thermistor will only be operated at high resistance values, but at lower resistance values it is recommended to use a four-wire circuit.

Figure 13: Four-wire circuit

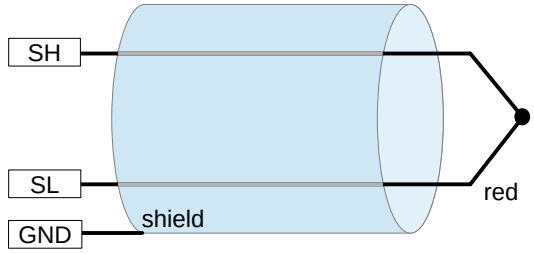


2.5.3 Thermocouples

Thermocouples must be connected to the SH and SL terminals; the PH and PL terminals must be left unconnected.

Connect the positive thermocouple wire to the SH terminal and the negative wire to the SL terminal. The insulation on thermocouple wire is usually color coded to indicate polarity, with red indicating the negative thermocouple wire.

Figure 14: Thermocouple connection

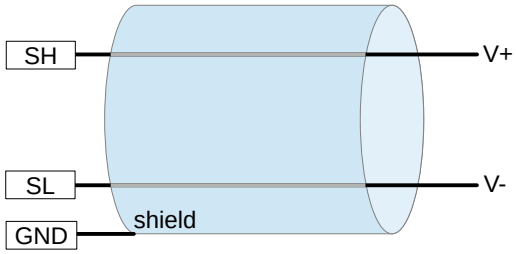


2.5.4 Voltage

When measuring a DC voltage, the voltage source must be connected to the SH and SL terminals and the PH and PL terminals must be left unconnected.

High common-mode voltage (CMV) can cause measurement errors or damage circuitry on the 2218. To avoid excessive CMV, it is recommended that you reference your voltage source to the GND terminal on the 2218 terminal block. If necessary, use a dedicated signal GND conductor to do this as shown in Figure 11.

Figure 15: DC voltage connection



2.6 Measurement noise

2.6.1 Warm-up noise

Sensor data may be noisy when the module is warming up or subjected to thermal transients. This is characterized by sensor data that periodically “jumps” and then drifts for approximately seven seconds.

Warm-up noise will subside when the module reaches thermal stability. For best accuracy, it is recommended to allow time for the module to reach thermal stability and to protect it from sudden temperature changes.

2.6.2 Other noise

It is not within the scope of this manual to discuss all causes and treatments of noise, however, a few simple techniques are available which will solve many noise problems:

- The apparent noise may be the result of incorrect wiring or unanticipated external influences. Inspect the sensor signals with an oscilloscope to confirm they match your expectations.
- In the case of thermocouples and DC voltage sources, the sensor signals must be referenced to the module's GND terminal. If this rule is violated, external fields could induce voltage noise that the module cannot reject. In the case of thermocouples it may be possible to resolve this by grounding the hot junction, or in the case of a DC voltage source, by grounding either SH or SL (but not both).

Chapter 3: API

3.1 Overview

The 2218 module is controlled and monitored by calling functions in the 2218 application program interface (API). Many of the API functions will cause the computer to send a command to the module, which in turn will cause the module to send a reply to the computer. All such functions return when the reply is received, or when a time limit has been exceeded, whichever occurs first.

3.1.1 Error codes

Every API function returns an error code that indicates whether it executed normally or, if not, what went wrong. The symbolic names of error codes are listed below; the numeric values of these codes can be found in the API source code.

Error code	Description
S2218_ERR_OK	Success – no errors detected.
S2218_ERR_OPCODE	Command not recognized.
S2218_ERR_ARG	Illegal function argument value.
S2218_ERR_OFFLINE	Analog circuitry is unpowered, but the function requires it to be powered.
S2218_ERR_FAULT	Failed to execute command due to hardware fault.
S2218_ERR_REOPEN	Attempted to power-up analog circuitry when it is already powered.
S2218_ERR_CREATE_RESOURCE	Failed to create system resources required for a 2218 module.
S2218_ERR_RESOURCE_CLOSED	A resource is closed or unavailable.
S2218_ERR_DEVICE_CLOSED	The module is closed, but the function requires it to be open.
S2218_ERR_DEVICE_OPEN	The function tried to open a 2218 module which is already open.
S2218_ERR_WAS_SHUTDOWN	The USB connection is closed.
S2218_ERR_READ	USB read error.
S2218_ERR_WRITE	USB write error.
S2218_ERR_OPENHIDLIB	Failed to open the system library HIDLIB.
S2218_ERR_DEVICE_NOT_FOUND	The system failed to detect the specified 2218 module.
S2218_ERR_COMMAND_MISMATCH	Message synchronization error between system and 2218 module.
S2218_ERR_API_OPEN	Attempted to open the API when it is already open.
S2218_ERR_API_CLOSED	Attempted to close the API when it is already closed.
S2218_ERR_LOCK_TIMEOUT	Failed to access a 2218 module because another thread/process is hogging it.
S2218_ERR_SIG_TIMEOUT	Timed out waiting for a reply from a 2218 module.
S2218_ERR_NOT_ALLOWED	Command prohibited by current settings.
S2218_ERR_SYSTEM	System error.

3.2 Admin functions

3.2.1 s2218_OpenApi

```
int s2218_OpenApi(int *devflags);
```

Arguments

devflags
Receives bit flags that indicate detected 2218 modules.

Description

This function initializes the API and detects all 2218 modules. It must be called once before any other API functions are called.

If the function succeeds, `devflags` will contains a set of bit flags indicating all detected modules. Each bit position corresponds to the the address programmed onto a module's rotary device address switch. For example, bit 0 will be set if a module with address 0 is detected. `devflags` will contain zero if no boards are detected.

If two or more modules have been assigned the same address, only one of the modules will be detected and the other modules will be inaccessible. Duplicate addresses are not reported by the API, so users must ensure that each module has been assigned a unique address.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
// Open the API and list all detected modules

int addr, flags, errcode = s2218_OpenApi(&flags);
if (errcode != S2218_ERR_OK)
  printf("s2218_OpenApi() returned error code %d", errcode);
else if (flags == 0)
  printf("No modules were detected");
else {
  printf("Modules were detected at these addresses:\n");
  for (addr = 0; addr < 16; addr++) {
    if (flags & (1 << addr))
      printf("%d\n", addr);
  }
}
```

3.2.2 s2218_CloseApi

`int s2218_CloseApi(void);`

Description

This function closes all open modules and then closes the API. It must be called once before the application program closes. No other API functions should be called after this function is called.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
int errcode = s2218_CloseApi();
```

3.2.3 s2218_OpenDevice

`int s2218_OpenDevice(int devaddr, int *online);`

Arguments

devaddr

Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

online
Receives module's online status.

Description

This function enables communication with a 2218 module. It must be called once for each 2218 module before calling other API functions that reference the module.

If the module was successfully opened, a status code is returned in `online`. The status code will be `S2218_ERR_OK` if the module is operating normally, or a non-zero error code if the module is operating abnormally.

Return value

The function returns `S2218_ERR_OK` if the module was opened, or a non-zero error code if the module was not opened.

Example

```
int online, errcode = s2218_OpenDevice(0, &online);
if (errcode != S2218_ERR_OK)
  printf("Can't open module: error code = %d", errcode);
else if (online != S2218_ERR_OK)
  printf("Module is operating abnormally: status code = %d", online);
else
  printf("Module is open and operating normally");
```

3.2.4 s2218_CloseDevice

`int s2218_CloseDevice(int devaddr);`

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

Description

This function must be called once for each 2218 module to terminate communication with the module. After calling this function, the application is not allowed to call any other API functions that reference the module.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
int errcode = s2218_CloseDevice();
if (errcode != S2218_ERR_OK)
  printf("s2218_CloseDevice() returned error code %d", errcode);
```

3.2.5 s2218_ErrString

`const char *s2218_ErrString(int errcode);`

Arguments

errcode
API error code.

Description

This function maps an API error code into a corresponding text string. It is useful for generating a descriptive error message when an API function returns an error code.

Return value

This function returns a pointer to a text string that describes the error associated with `errcode`.

Example

```
int errcode = s2218_CloseApi();
if (errcode != S2218_ERR_OK)
  printf("s2218_CloseApi() failed: %s\n", s2218_ErrString(errcode));
```

3.2.6 s2218_GetApiVersion

```
int s2218_GetApiVersion(S2218_VERSION *ver);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

ver
Buffer for API version number.

Description

This function copies the API version number to `ver`.

Return value

This function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
S2218_VERSION ver;
int errcode = s2218_GetApiVersion(&ver);
if (errcode != S2218_ERR_OK)
  printf(“ERROR!”);
else
  printf(“API version = %d.%d.%d\n”, ver.Major, ver.Minor, ver.Build);
```

3.2.7 s2218_GetDeviceVersions

```
int s2218_GetDeviceVersions(int devaddr, S2218_VERSION_INFO *ver);
```

Arguments

devaddr
Device address. This must match the settings of the 2218 module's rotary switch as described in section 2.2.1.

ver
Buffer for device version information.

Description

This function copies device version information to `ver`.

Return value

This function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
S2218_VERSION_INFO ver;
int errcode = s2218_GetDeviceVersions(&ver);
if (errcode != S2218_ERR_OK)
  printf("ERROR!");
else {
  printf("Dev version %d.%d.%d\n", ver.DevFw.Major, ver.DevFw.Minor, ver.DevFw.Build);
  printf("Smad version %d.%d.%d\n", ver.SmadFw.Major, ver.SmadFw.Minor, ver.SmadFw.Build);
  printf("PWB rev %c", ver.PwbRev);
}
```

3.3 Analog configuration

3.3.1 s2218_SetSensorType

```
int s2218_SetSensorType(int devaddr, int chan, SENSOR_TYPE sdc);
```

Arguments

devaddr

Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

chan

Sensor channel number in the range [0:7].

sdc

Enumerated sensor type. Symbolic names for the sensor type codes are listed below; the associated numeric values can be found in the API source code:

Sensor type code	Sensor attributes		
	Class	Type/range	Data units
SDC_DISABLED	Disabled sensor channel		
SDC_VOLTS_5V	Voltage	5 V range	Volts
SDC_VOLTS_500MV		500 mV range	
SDC_VOLTS_100MV		100 mV range	
SDC_OHMS_400	Resistance	400 Ω range	Ω
SDC_OHMS_4K		4 kΩ range	
SDC_OHMS_600K		600 kΩ range	
SDC_TEMP_THERMOCOUPLE_B	Thermocouple	Type B	°C
SDC_TEMP_THERMOCOUPLE_C		Type C	
SDC_TEMP_THERMOCOUPLE_E		Type E	
SDC_TEMP_THERMOCOUPLE_J		Type J	
SDC_TEMP_THERMOCOUPLE_K		Type K	
SDC_TEMP_THERMOCOUPLE_N		Type N	
SDC_TEMP_THERMOCOUPLE_T		Type T	
SDC_TEMP_THERMOCOUPLE_S		Type S	
SDC_TEMP_THERMOCOUPLE_R		Type R	
SDC_TEMP_THERMISTOR	Thermistor	10K	
SDC_TEMP_RTD_PT100_385_400	RTD	Platinum, α = 385, 400 °C range	

SDC_TEMP_RTD_PT100_385_800		Platinum, α = 385, 800 °C range	
SDC_TEMP_RTD_PT100_392_400		Platinum, α = 392, 400 °C range	
SDC_TEMP_RTD_PT100_392_800		Platinum, α = 392, 800 °C range	
SDC_TEMP_RTD_NI200		Nickel, 200	
SDC_TEMP_RTD_NI1000		Nickel, 1000	
SDC_TEMP_RTD_CU10		Copper, 10	

Description

This function configures a sensor channel to measure a particular sensor type. Each invocation configures one channel, so the function must be called eight times to configure all channels. After calling this function, valid sensor data will not be available until the channel has been internally scanned.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
// Configure sensor channel 2 for type K thermocouple measurement
int errcode = s2218_SetSensorType(0, 2, SDC_TEMP_THERMOCOUPLE_K);
if (errcode != S2218_ERR_OK)
  printf(“ERROR!”);
```

3.3.2 s2218_SetOpenSensorValues

```
int s2218_SetOpenSensorValues(int devaddr, u8 failHighFlags);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

failHighFlags
Flag bits (one per channel) that specify the data value expected upon open-sensor detection: 1 = maximum value; 0 = minimum value.

Description

This function establishes the data values that the module will report in the event of an ADC over- or under-range. It applies to all sensor types but is especially useful for channels that have been configured for thermocouples (via `s2218_SetSensorType`), as they employ special circuitry to facilitate open thermocouple detection via forced ADC over-range. This can be leveraged to trigger alarms when open sensors are detected, and to force the desired system responses to fault conditions in closed-loop process control applications.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
// Program channels 0-3 to fail high, channels 4-7 to fail low.
int errcode = s2218_SetOpenSensorValues(0, 0x0F);
if (errcode != S2218_ERR_OK)
  printf(“ERROR!”);
```

3.3.3 s2218_GetAnalogConfig

```
int s2218_GetAnalogConfig(int devaddr, S2218_ANALOG_CONFIG *cfg);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

cfg
Buffer for analog configuration settings.

Description

This function returns the module's analog configuration settings in `cfg`, which must be large enough to accommodate a `S2218_CHAN_CONFIG` structure.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
// Display module's analog configuration.
int i;
S2218_ANALOG_CONFIG cfg;
S2218_CHAN_CONFIG *ch = cfg.Chan;
int errcode = s2218_GetAnalogConfig(0, &cfg);
if (errcode != S2218_ERR_OK)
  printf("ERROR!");
else {
  printf("Sensor scan period = %d ms\n", cfg.ScanPeriod);
  printf("Send data report rate every %d scan(s)\n", cfg.DataReportRate);
  printf("CHAN, TYPE, OPEN, HILIM, LOLIM:\n");
  for (i = 0; i < S2218_NUM_CHANS; ch++, i++) {
    printf("%d,", i);
    printf("%d,", ch->SensorType);
    printf("%s,", (cfg.OpenFlags >> i) & 1 ? "High" : "Low");
    if (ch->AlarmLimitHigh.Enable) printf("%d,", ch->AlarmLimitHigh.Value); else printf("Disabled");
    if (ch->AlarmLimitLow.Enable)  printf("%d,", ch->AlarmLimitLow.Value);  else printf("Disabled");
  }
}
```

3.4 Data acquisition

3.4.1 Data reports

The 2218 module sends sensor data to your computer via messages called *data reports*. A data report consists of eight sensor samples (one per channel), a timestamp that indicates when the message was enqueued for transmission, and various other information. To ensure valid sensor data, you must first enable sensor scanning by calling `s2218_SetTrigConfig`, and then enable data report transmissions by calling `s2218_SetTrigEnable`.

The API collects received data reports in a FIFO buffer so that your software can process them at convenient times. The FIFO capacity is user-configurable at run time. If a new report arrives while the FIFO is full, it (or the oldest report in the FIFO, depending on your preferred policy) will be dropped. Each report includes the total number of dropped reports since the module booted; this is the only indication of dropped reports. Call `s2218_WaitForDataReport` to receive the oldest report in the FIFO.

3.4.2 Stream control

Two control modes are available for data report streaming: *triggered* and *periodic*. In both modes, the module will send a data report upon user-selectable GPIO edge (configured via `s2218_SetGpioConfig`). In the periodic mode the module will also transmit data reports according to a user-defined, periodic schedule.

Data streaming is disabled by default when the module is powered up. When your program is ready to start receiving sensor data, it must call `s2218_SetTrigEnable` to select the stream control mode and enable streaming. This will enable GPIO data report triggering and, in the timer mode, also start periodic data reports. When data reports are no longer needed, the program should call `s2218_GetTrigEnable` to terminate the stream.

When using a GPIO to trigger data reports, the GPIO pin may be driven by an external signal or by its own output driver. To synchronize data reports to an external signal (e.g., sampling clock), configure the GPIO pin as an input and connect it to the external signal. Alternatively, the GPIO pin may be driven by the GPIO itself, which allows your software to trigger a data report on demand. To implement this, configure the pin as an output and call `s2218_SetGpioOutputs` to trigger a data report.

3.4.3 Functions

3.4.3.1 s2218_SetTrigConfig

```
int s2218_SetTrigConfig(int devaddr, S2218_TRIG_CONFIG *cfg);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

cfg
S2218_TRIG_CONFIG structure.

Description

This function configures the module's internal scanning parameters. See api2218.h for details. It includes the periodic report rate (scans per report) and GPIO edge selector flags (see Data reports for details).

When the module performs a scan, it acquires and caches eight samples (one sample from each sensor channel) and compares each sample to its programmed alarm limits. An alarm notification will be immediately sent to the computer via s2218_WaitForLimitAlarm if a limit violation is detected and alarm reports are enabled. The cached samples are used whenever a data report is generated.

The scanning period is specified by `period`. For example, when `period=500`, the module will perform one scan every 500 ms, which corresponds to two scans per second. The minimum period is 200 ms; the function will fail and return `CMD2218_ERR_ARG` if `period` is less than 200.

It is not recommended to call this function while data reports are streaming. If it's necessary to change the scan period while streaming, stop the stream (via `s2218_SetTrigEnable`) before calling this function and then restart the stream after this function returns.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

3.4.3.2 s2218_GetTrigConfig

```
int s2218_GetTrigConfig(int devaddr, S2218_TRIG_CONFIG *cfg);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

cfg
S2218_TRIG_CONFIG structure.

Description

This function returns the current trigger configuration.

3.4.3.3 s2218_SetTrigEnable

```
int s2218_SetTrigEnable(int devaddr, u8 enable);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

enable
Enable data acquisition.

Description

This function enables data report streaming and creates a FIFO buffer to receive data reports.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected. The function will fail and return `S2218_ERR_NOT_ALLOWED` if streaming is already enabled.

3.4.3.4 s2218_GetTrigEnable

```
int s2218_GetTrigEnable(int devaddr, u8 *enable);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

enable
Current value (pointer to) of data acquisition enable

Description

This function retrieves the status of trigger enable.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

3.4.3.5 s2218_WaitForDataReport

```
int s2218_WaitForDataReport(int devaddr, S2218_DATAREPORT *report, int tmax);
```

Arguments

devaddr

Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

report

Buffer for the data report.

tmax

Maximum time to wait for a data report, in milliseconds.

Description

This function fetches the next available report from the data report FIFO and stores it in `report`. It will return immediately if the FIFO is not empty or `tmax=0`, otherwise it will block until a data report arrives or `tmax` has elapsed.

The function supports both polled and event-driven applications. For polled (non-blocking) operation, set `tmax=0`; this will cause the function to return immediately even if the FIFO is empty. For event-driven applications, it is recommended to set `tmax=S2218_INFINITE_WAIT`; this will cause the function to unconditionally block until a report arrives or an error is detected.

Return value

The function will return `S2218_ERR_OK` if a data report is received within `tmax`. If `tmax` elapses before a data report becomes available, or if `tmax=0` and the FIFO is empty, the function will return `S2218_ERR_SIG_TIMEOUT`.

The function will return `ERR_RESOURCE_CLOSED` if data report streaming is disabled. Also, if another thread disables streaming (by calling `s2218_GetTrigEnable`) while a caller is waiting in this function, the wait will be canceled and this function will immediately return `ERR_RESOURCE_CLOSED`.

Example

```
// Event-driven operation: block until report is available
S2218_DATAREPORT report;
int errcode = s2218_WaitForDataReport(0, &report, S2218_INFINITE_WAIT); // block until report arrives
switch (errcode) {
  case S2218_ERR_OK:              DisplayDataReport(&report); break;
  case S2218_ERR_RESOURCE_CLOSED: printf(“Stream was halted by another thread\n”); break;
  default:                        printf("%s\n", s2218_ErrString(errcode));
}
```

```
// Polled operation: return immediately even if no report is available
S2218_DATAREPORT report;
int errcode = s2218_WaitForDataReport(0, &report, 0); // never block
switch (errcode) {
  case S2218_ERR_OK:              DisplayDataReport(&report); break;
  case S2218_ERR_SIG_TIMEOUT:     printf(“No report available\n”); break;
  case S2218_ERR_RESOURCE_CLOSED: printf(“Stream was halted by another thread\n”); break;
  default:                        printf("%s\n", s2218_ErrString(errcode));
}
```

```
// Display samples from sensor channels 2 and 5, which are assumed to be configured for thermocouples.
void DisplayDataReport(S2218_DATAREPORT *report)
{
  double timestamp = report->TstampSec + report->TstampMsec / 1000.0;  // timestamp in seconds
  printf(“Data received at timestamp = %f10.1:\n”, timestamp);
  printf(“ chan 2 = %f degrees C\n”, report->Data[2]);
  printf(“ chan 5 = %f degrees C\n”, report->Data[5]);
}
```

3.4.3.6 s2218_ReadDeviceTemp

```
int s2218_ReadDeviceTemp(int devaddr, double *degC);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

degC
Buffer for temperature in degrees C.

Description

This function reads the module's internal temperature and stores it in `degC`.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
double degC;
int errcode = s2218_ReadDeviceTemp(0, &degC);
if (errcode != S2218_ERR_OK)
  printf("ERROR!");
else
  printf("Module temperature = %f degrees C\n", degC);
```

3.5 Alarms

A 2218 module can automatically monitor sensor data and notify your software when a sensor's data value strays outside user-defined limits. To enable this function, your application program must first call `s2218_SetTrigConfig` and s2218_SetTrigEnable to configure the module's internal scan rate and start it scanning. The scan rate determines how frequently the module will check for limit violations.

Call s2218_SetLoThreshold and `s2218_SetHiThreshold` to configure a channel's upper or lower limit thresholds. Together, the upper and lower limits specify the normal data range of the sensor. Call s2218_SetLoLimitEnable and s2218_SetHiLimitEnable to enable limit reports without alarm trigger. Call s2218_SetLoAlarmEnable and/or s2218_SetHiAlarmEnable to enable alarm reporting.

An alarm will "sound" when a limit violation is detected (i.e., sensor data value is greater than a channel's upper limit or less than its lower limit). When this happens, the channel's high and low alarms are both automatically disabled and an alarm report is sent to the computer. Use s2218_WaitForLimitAlarm to wait for and receive the report.

Every alarm report includes a timestamp that indicates when the limit violation was detected and a set of bit flags that indicate which limits were violated:

Flags	High alarms								Low alarms							
Bit number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sensor channel	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

The API stores alarm reports in a FIFO so that the application program won't miss rapid back-to-back alarm reports. To receive alarm notifications, the application must call `s2218_WaitForLimitAlarm`, which returns the oldest report in the FIFO. Upon FIFO overflow, the oldest alarm report will be dropped from the FIFO to make space for the newest report. Each alarm report includes the total number of dropped reports since the module booted; this is the only indication of dropped alarm reports.

3.5.1 s2218_SetLoThreshold

```
int s2218_SetLoThreshLimit(int devaddr, u8 chan, double limit);
```

Arguments

devaddr

Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

chan

Sensor channel number in the range [0:7].

limit

Data limit. The most negative data value in the normal operating range.

Description

This function configures a sensor channel's lower limit alarm threshold.

The `limit` value is expressed in the data units used by the channel's sensor type (e.g., Volts, ohms, °C). For example, `limit` is expressed in °C for a channel that is measuring thermocouples.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
// Enable lower alarm limit of 375 degrees C on a thermocouple measurement channel.
int errcode = s2218_SetLoThreshold(0, 0, 375);
if (errcode != S2218_ERR_OK)
  printf(“ERROR!”);
```

3.5.2 s2218_SetHiThreshold

```
int s2218_SetHiThreshold(int devaddr, u8 chan, double limit);
```

Arguments

devaddr

Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

chan

Sensor channel number in the range [0:7].

limit

Data limit. The most positive data value in the normal operating range.

Description

This function configures a sensor channel's upper limit alarm threshold.

The `limit` value is expressed in the data units used by the channel's sensor type (e.g., Volts, ohms, °C). For example, `limit` is expressed in °C for a channel that is measuring thermocouples.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

3.5.3 s2218_SetLoLimitEnable

```
int s2218_SetLoLimitEnable(int devaddr, u8 chan, int enable);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

chan
Sensor channel number in the range [0:7].

enable
Limit enable: 1 = enable limit reports for this limit; 0 = disable limit reports for this limit.

Description

This function configures whether a sensor channel reports limit events in data reports.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

3.5.4 s2218_SetHiLimitEnable

```
int s2218_SetHiLimitEnable(int devaddr, u8 chan, int enable);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

chan
Sensor channel number in the range [0:7].

enable
Limit enable: 1 = enable limit reports for this limit; 0 = disable limit reports for this limit.

Description

This function configures whether a sensor channel reports limit events in data reports.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

3.5.5 s2218_SetLoAlarmEnable

```
int s2218_SetLoAlarmEnable(int devaddr, u8 chan, int enable);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

chan
Sensor channel number in the range [0:7].

enable
Alarm enable: 1 = enable alarm for this limit; 0 = disable alarm for this limit.

Description

This function enables or disables the lower limit alarm. s2218_WaitForLimitAlarm will return when an alarm occurs.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

3.5.6 s2218_SetHiAlarmEnable

```
int s2218_SetHiAlarmEnable(int devaddr, u8 chan, int enable);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

chan
Sensor channel number in the range [0:7].

enable
Alarm enable: 1 = enable alarm for this limit; 0 = disable alarm for this limit.

Description

This function enables or disables the upper limit alarm. s2218_WaitForLimitAlarm will return when an alarm occurs.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

3.5.7 s2218_WaitForLimitAlarm

```
int s2218_WaitForLimitAlarm(int devaddr, S2218_ALARM_REPORT *report, int tmax);
```

Arguments

devaddr
Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

report
Buffer for alarm report.

tmax
Maximum time to wait for alarm report in milliseconds.

Description

This function fetches the oldest alarm report from the alarm FIFO and stores it in `report`. It will return immediately if a report is available in the FIFO or `tmax=0`, otherwise it will block until a report arrives or `tmax` has elapsed.

To implement non-blocking (i.e., polled) operation, set `tmax=0`; this will cause the function to return immediately even if the FIFO is empty. To disable timeouts, set `tmax=S2218_INFINITE_WAIT`; this will cause the function to block until a report is available in the FIFO.

Return value

The function will return `S2218_ERR_OK` if a report was successfully fetched. If `tmax` elapses before a report becomes available, or if `tmax=0` and the alarm FIFO is empty, the function will return `S2218_ERR_SIG_TIMEOUT`.

If another thread closes the module (by calling `s2218_CloseDevice`) while a caller is waiting in this function, the wait will be canceled and this function will immediately return `ERR_DEVICE_CLOSED`.

Examples

```
// Event-driven operation: Block until a report is available in the alarm FIFO
S2218_ALARMREPORT report;
int errcode = s2218_WaitForLimitAlarm(0, &report, S2218_INFINITE_WAIT);  // Wait for report
if (errcode == S2218_ERR_OK)
  ProcessAlarms(&report);
else
  printf("ERROR!");
```

```
// Polled operation: Fetch report if one is available, but don't block if alarm FIFO is empty
S2218_ALARMREPORT report;
int errcode = s2218_WaitForLimitAlarm(0, &report, 0);  // Check for alarms without blocking
if (errcode == S2218_ERR_OK)                // if alarm report was fetched
  ProcessAlarms(&report);
else if (errcode == S2218_ERR_SIG_TIMEOUT)  // else if alarm FIFO is empty
  printf("No alarms sounding");
else                                        // else must be an error
  printf("ERROR!");
```

```
// Display alarm report
void ProcessAlarms(S2218_ALARMREPORT *report)
{
  int chan;
  u16 flags = report->Event.Flags;
  u16 mask = 1;
  printf("Alarm reported at time = %d.%3d", report->Event.Timestamp.s, report->Event.Timestamp.ms);
  for (chan = 0; chan < 8; chan++, mask <<= 1)
    if (flags & mask) printf("channel %d low alarm\n", chan);
  for (chan = 0; chan < 8; chan++, mask <<= 1)
    if (flags & mask) printf("channel %d high alarm\n", chan);
}
```

3.6 GPIO

3.6.1 Data structures

3.6.1.1 Flag byte

Several GPIO-related structures contain flag bytes. A flag byte consists of eight bit flags, wherein each bit is associated with a particular GPIO:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
GPIO	GPIO7	GPIO6	GPIO5	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0

3.6.1.2 S2218_GPIO_PAIR

This structure contains two bytes which are associated with rising and falling GPIO edges, respectively. Each byte may be an integer value or a flag byte.

`u8 Rising`

Integer value or flag byte associated with rising GPIO edges.

`u8 Falling`

Integer value or flag byte associated with falling GPIO edges.

3.6.1.3 S2218_GPIO_CONFIG

This structure specifies the operating modes of all GPIOs.

`S2218_GPIO_PAIR DbTime[8]`

Debounce intervals expressed in milliseconds. The array index is the GPIO number. Each GPIO has two debounce intervals, one for rising edges and one for falling edges. Each interval may be assigned an integer value in the range [0:255]; set to 0 to disable debounce. Example: `DbTime[3].Falling=20` will apply 20 ms debounce to all falling edges on GPIO3.

`u8 DataTrigSelect`

Configure hardware trigger for data reports.

EN	0	0	0	R/$\overline{F}$	SEL2	SEL1	SEL0

EN Trigger enable: 1 = enable; 0 = disable.

R/$\overline{F}$ Edge type: 1= rising; 0 = falling.

SEL GPIO identifier in range [0:7].

This configures a GPIO to act as a data report trigger. When the specified edge is detected, the module will send a data report over USB. For example: set `DataTrigSelect=0x88+4` to send data reports upon rising edges of GPIO4. Note: When using a GPIO to trigger data reports, it is recommended to allow at least 200 ms between consecutive trigger edges.

`u8 PinDirections`

Pin direction control flags: 1=input; 0=output. It is recommended to program unused pins as outputs.

`u8 BiasEnables`

Pin bias enable flags: 1=enable bias resistor; 0=disable bias resistor. A flag will be ignored if the corresponding GPIO is an output.

`u8 BiasPolarities`

Pin bias voltage control: 1=pull-up to +3.3V; 0=pull-down to 0V. A flag will be ignored if the corresponding GPIO is an output or its pin bias is disabled.

3.6.1.4 S2218_GPIO_SETTINGS

This structure indicates the operating modes and control states of all GPIOs.

`S2218_GPIO_CONFIG Config`

Operating modes of all GPIOs.

`u8 OutputStates`

Output state flags. These flags indicate the programmed logic states of all GPIOs. Note: A flag will indicate the pin's physical output condition only if the GPIO is an output.

`S2218_GPIO_PAIR CapEnables`

Edge capture enable flags. These flags indicate, for each edge type of each GPIO, whether event capturing is enabled for that edge type.

3.6.2 Functions

3.6.2.1 s2218_SetGpioOutputs

```
int s2218_SetGpioOutputs(int devaddr, u8 states);
```

Arguments

devaddr

Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

states

Bit flags (one bit per GPIO) which indicate the desired output levels of all GPIOs: 1=output high (3.3 V); 0=output low (0 V).

Description

This function programs the output voltage levels on all GPIOs. Note that it does not affect the voltage levels on GPIO pins that are configured as inputs.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
// Program GPIO7 and GPIO1 to 3.3V, all others to 0V
int errcode = s2218_SetGpioOutputs(0, 0x82);
if (errcode != S2218_ERR_OK)
  printf(“ERROR!”);
```

3.6.2.2 s2218_GetGpioInputs

```
int s2218_GetGpioInputs(int devaddr, u8 *states);
```

Arguments

devaddr

Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

states

Buffer to receive bit flags (1 bit per GPIO) which indicate the logic levels detected at all GPIO pins: 1=high (3.3 V); 0=low (0 V).

Description

This function returns the debounced logic levels detected at the GPIO pins. The logic levels of the GPIOs are stored in `states`.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
// Read and display GPIO pin states
u8 states;
int errcode = s2218_GetGpioInputs(0, &states);
if (errcode != S2218_ERR_OK)
  printf(“ERROR!”);
else {
  int gpio;
  for (gpio = 0; gpio < 8; gpio++, states >>= 1)
    printf(“GPIO %d state = %d\n”, gpio, states & 1);
}
```

3.6.2.3 s2218_SetGpioConfig

```
int s2218_SetGpioConfig(int devaddr, S2218_GPIO_CONFIG *cfg);
```

Arguments

devaddr

Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

cfg

Configuration settings for all GPIOs. See section 3.6.1.3 for details.

Description

This function programs the operating modes of all GPIOs. Before calling this function, the `cfg` structure must be filled with the desired configuration settings.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
// Configure all GPIOs
S2218_GPIO_CONFIG cfg = {0,};       // Create config struct and zero all members.
cfg.PinDirections = 0x07;           // Config GPIO-GPIO2 as inputs; others as outputs.
cfg.BiasEnables = 0x06;             // Enable bias resistors on GPIO1 and GPIO2.
cfg.BiasPolarities = 0x04;          // Pull-up resistor on GPIO2; pull-down on GPIO1.
cfg.DbTime[2].Rising = 20;          // GPIO2 rising edge debounce = 20 ms.
cfg.DataTrigSelect = 0x82;          // Use GPIO2 falling edges to trigger data reports.

if (s2218_SetGpioConfig(0, &cfg) != S2218_ERR_OK)   // Activate the configuration
  printf(“ERROR!”);
```

3.6.2.4 s2218_GetGpioSettings

```
int s2218_GetGpioSettings(int devaddr, S2218_GPIO_SETTINGS *cfg);
```

Arguments

devaddr

Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

cfg

Buffer for GPIO configuration and control state information. See section 3.6.1.4 for details.

Description

This function reads the configuration modes and control states of all GPIOs and copies them to `cfg`.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
// Read and display GPIO capture enables
S2218_GPIO_SETTINGS cfg;
if (s2218_GetGpioSettings(0, &cfg) != S2218_ERR_OK)
  printf("ERROR!");
else {
  int gpio;
  for (gpio = 0; gpio < 8; gpio++)
    printf("GPIO%d capture enables: RisingEdge=%d FallingEdge=%d\n", gpio,
      (cfg.CapEnables.Rising >> gpio) & 1,
      (cfg.CapEnables.Falling >> gpio) & 1);
}
```

3.7 Edge capture

The module can automatically detect debounced GPIO edges and, when an edge event is detected, send a GPIO event report to your computer. Every report includes a timestamp that indicates when the event occurred and bit flags that indicate which edges were detected. Call `s2218_GpioCapEnable` to selectively enable or disable event capturing for all GPIOs. To read the current capture enables of all GPIOs, call `s2218_GetGpioSettings`.

The API stores GPIO event reports in a FIFO buffer so that the application program won't miss rapid back-to-back reports. To receive event notifications, the application must call `s2218_WaitForGpioEvent`, which returns the oldest report in the FIFO. Upon FIFO overflow, the oldest report will automatically be dropped from the FIFO to make space for the newest report. Each report includes the total number of dropped reports since the module booted; this is the only indication of dropped reports.

3.7.1.1 s2218_GpioCapEnable

```
int s2218_GpioCapEnable(int devaddr, S2218_GPIO_PAIR *enables);
```

Arguments

devaddr

Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

enables

Edge capture enable flags: 1=enable; 0=disable.

Description

This function configures the capture enables of all GPIO edges. Before calling the function, `enables` must be filled with flags that specify the capture enable for all edges. For each edge, set the corresponding flag to '1' to enable capturing or '0' to disable capturing.

Return value

The function returns `S2218_ERR_OK` if successful, or a non-zero error code if a problem was detected.

Example

```
// Enable capturing of GPIO2 rising edges and GPIO2-GPIO4 falling edges
// and disable capturing on all other edges
S2218_GPIO_PAIR enables;    // gpio 76543210
enables.Rising = 0x04;      //      00000100
enables.Falling = 0x1C;     //      00011100
if (s2218_GpioCapEnable(0, &enables, 1) != S2218_ERR_OK)
  printf("ERROR!");
```

3.7.1.2 s2218_WaitForGpioEvent

```
int s2218_WaitForGpioEvent(int devaddr, S2218_GPIOREPORT *report, int tmax);
```

Arguments

devaddr

Device address. This must match the settings of the module's rotary switch as described in section 2.2.1.

report

Buffer for received GPIO event report.

tmax

Maximum time to wait for the report, in milliseconds.

Description

This function fetches the oldest report from the GPIO event FIFO and stores it in `report`. It will return immediately if a report is available in the FIFO or `tmax=0`, otherwise it will block until a report arrives or `tmax` has elapsed.

To implement non-blocking (i.e., polled) operation, set `tmax=0`; this will cause the function to return immediately even if the FIFO is empty. To disable timeouts, set `tmax=S2218_INFINITE_WAIT`; this will cause the function to block until a report is available in the FIFO.

Return value

The function will return `S2218_ERR_OK` if a report was successfully fetched. If `tmax` elapses before a report becomes available, or if `tmax=0` and the FIFO is empty, the function will return `S2218_ERR_SIG_TIMEOUT`.

If another thread closes the module (by calling `s2218_CloseDevice`) while a caller is waiting in this function, the wait will be canceled and this function will immediately return `ERR_DEVICE_CLOSED`.

Example

```
// Event-driven operation: Block until a report is available in the FIFO.
S2218_GPIOREPORT report;
int errcode = s2218_WaitForGpioEvent(0, &report, S2218_INFINITE_WAIT);
if (errcode == S2218_ERR_OK)
  DisplayGpioEventReport(&report);
else
  printf("ERROR!");
```

```
// Polled operation: Fetch report if one is available, but don't block if FIFO is empty
S2218_GPIOREPORT report;
int errcode = s2218_WaitForGpioEvent(0, &report, 0);  // Check for events without blocking
if (errcode == S2218_ERR_OK)                // if alarm report was fetched
  DisplayGpioEventReport(&report);
else if (errcode == S2218_ERR_SIG_TIMEOUT)  // else if FIFO is empty
  printf("No GPIO events captured");
else                                        // else must be an error
  printf("ERROR!");
```

```
// Display a GPIO event report.

void DisplayGpioEventReport(S2218_GPIOREPORT *report)
{
  int gpio;
  u16 mask = 1;

  printf("GPIO edges at time = %d.%3d:\n", report->Event.Timestamp.s, report->Event.Timestamp.ms);

  for (gpio = 0; gpio < 8; gpio++, mask <<= 1) {
    if (report.Flags.Rising  & mask) printf("gpio%d rising edge\n", gpio);
    if (report.Flags.Falling & mask) printf("gpio%d falling edge\n", gpio);
  }
}
```

Chapter 4: Specifications

4.1.1 General specifications

Parameter			Specification
Sensor interface	Channels	Number	8
		Sensor type	Configurable per channel
	ADC	Type	Integrating
		Conversion time	16.67 ms, nominal
	Sampling rate	Aggregate	45 samples/s
		Per channel	5.6 samples/s (all channels active)
	Input	Type	Differential voltage
		Differential voltage range	-5 to +5 V
		CMV range	-5 to +5 V
		CMRR	78 dB @ ≤ 60 Hz, min
		Absolute maximum voltage	±20 V
		Input impedance	100 MΩ, nominal
	Excitation (pulsed)	RTD / 400 Ω range	1.3 mADC
		Thermistor / other Ω ranges	5 VDC || 4 KΩ
GPIO	Channels	Number	8
		Type	3.3 V logic, non-isolated
		Pin directions	Configurable per channel
	Power output (VCC33)	Voltage	+3.3 V ±3%
		Current [1]	20 mA maximum
	Input	Debounce interval	Configurable per channel, 0 to 255 ms in 1 ms steps
		Edge detection	1 ms minimum pulse width
		Voltage range	0 to VCC33 (not 5 V tolerant)
		Logic '0' voltage	0.7 V maximum
		Logic '1' voltage	2.25 V minimum
		Leakage current	±1 µA maximum (w/ pull-up and pull-down disabled)
		Pull-up resistance	13 KΩ typical @ 25 °C
		Pull-down resistance	66 KΩ typical @ 25 °C
	Output	Logic '0' voltage	0.4 V maximum
		Logic '1' voltage	2.4 V minimum
		Current [1]	10 mA maximum (source or sink)
System	Interface	Type	USB 2.0 high-speed
	Power	Source	USB
		Consumption	2 W maximum (400 mA @ 5 VDC)
	Temperature	Operating	0 to 70 °C
		Storage	-25 to 85 °C

Notes:
1. The sum of the currents flowing in VCC33 and GPIO pins must not exceed 20 mA.

4.1.2 Sensor specifications

Sensor			Range	Accuracy	Output data scaling/bit
Type		Code			
Disabled channel		0	n/a	n/a	n/a
DC voltage		1	±5 V	400 µV	200 µV
		2	±500 mV	30 µV	20 µV
		3	±100 mV	30 µV	5 µV
Resistance		4	0 to 400 Ω	0.04 Ω	0.02 Ω
		5	0 to 4 KΩ	0.25 Ω	0.125 Ω
		6	0 to 600 KΩ	130 Ω	31 Ω
Thermocouple	B	7	0 to 1820 °C	3.3 °C	0.1 °C
	C	8	0 to 1820 °C	2.1 °C	
	E	9	-270 to 990 °C	0.8 °C	
	J	10	-210 to 760 °C	0.6 °C	
	K	11	-270 to 1360 °C	1.0 °C	
	N	12	-270 to 1347 °C	0.9 °C	
	T	13	-270 to 400 °C	0.6 °C	
	S	14	0 to 1760 °C	3.0 °C	
	R	15	0 to 1760 °C	2.8 °C	
Thermistor	10 KΩ 44006 / 44031	16	-55 to 62 °C 62 to 126 °C 126 to 150 °C	0.01 °C 0.025 °C 0.008 °C	0.01 °C
RTD	Pt 100, α = 0.385	17 18	-200 to 400 °C -200 to 800 °C	0.2 °C 0.2 °C	0.0125 °F (~0.007 °C) 0.05 °C
	Pt 100, α = 0.392	19 20	-200 to 400 °C -200 to 800 °C	0.2 °C 0.2 °C	0.0125 °F (~0.007 °C) 0.05 °C
	Ni 200, α = 1.098	21	-60 to 180 °C	0.08 °C	0.05 °C
	Ni 1000, α = 4.4	22	-50 to 70 °C	0.02 °C	0.05 °C
	Cu 10, α = 0.039	23	0 to 125 °C	0.5 °C	0.1 °C
Current loop		25	4-to-20 mA	0.02 %	0.01 % (4 mA=0%, 20 mA=100%)

Chapter 5: Limited warranty

Sensoray Company, Incorporated (Sensoray) warrants the Model 2218 hardware to be free from defects in material and workmanship and perform to applicable published Sensoray specifications for two years from the date of shipment to purchaser. Sensoray will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

The warranty provided herein does not cover equipment subjected to abuse, misuse, accident, alteration, neglect, or unauthorized repair or installation. Sensoray shall have the right of final determination as to the existence and cause of defect.

As for items repaired or replaced under warranty, the warranty shall continue in effect for the remainder of the original warranty period, or for ninety days following date of shipment by Sensoray of the repaired or replaced part, whichever period is longer.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. Sensoray will pay the shipping costs of returning to the owner parts that are covered by warranty. A restocking charge of 25% of the product purchase price will be charged for returning a product to stock.

Sensoray believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, Sensoray reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition.

The reader should consult Sensoray if errors are suspected. In no event shall Sensoray be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, SENSORAY MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF SENSORAY SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. SENSORAY WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.