Hardware

INTRODUCTION
+ Concept
+ Architecture
+ konashi.js
+ Versions

SPECIFICATIONS
+ Supported devices
+ Appearance
+ Schematic

CORE FUNCTIONS
+ Digital
+ Analog
+ PWM
+ Communication
+ Bluetooth Low Energy
+ Event-driven

API Reference

CONSTANTS
+ Pin name
+ PIO
+ AIO
+ PWM
+ UART
+ I²C
+ Function return
+ Events

BASE
+ initialize
+ find
+ findWithName
+ disconnect
+ isConnected
+ peripheralName

EVENTS
+ addObserver
+ removeObserver

DIGITAL I/O (PIO)
**DIGITAL I/O (PIO)**
- pinMode
- pinModeAll
- pinPullup
- pinPullupAll
- digitalRead
- digitalReadAll
- digitalWrite
- digitalWriteAll

**ANALOG I/O (AIO)**
- analogReference
- analogReadRequest
- analogRead
- analogWrite

**PWM**
- pwmMode
- pwmPeriod
- pwmDuty
- pwmLedDrive

**UART**
- uartMode
- uartBaudrate
- uartWrite

**I²C**
- i2cMode
- i2cStartCondition
- i2cRestartCondition
- i2cStopCondition
- i2cWrite
- i2cReadRequest
- i2cRead

**HARDWARE CONTROL**
- reset
- batteryLevelReadRequest
- batteryLevelRead
- signalStrengthReadRequest
- signalStrengthRead

**JAVASCRIPT EVENT API**
- Abstract
- centralManagerPoweredOn
- peripheralNotFound
- connected
- disconnected
Physical-computing is a concept which envisions new interactions between people and computers in our everyday environment. It was first proposed by Dan O’Sullivan of New York University[1]. This concept is expected to make possible interactions which are beyond the limitations of current computer systems. For example, by measuring the movements of players, Kinect for Xbox360 changed the way people play games and experience entertainment.

But there are some large difficulties faced when trying to design a physical-computing interface. Developing a device based on the concept of physical computing is an undertaking comparable to recreating the input-output system of a current personal computer. To such facilitate innovation cross-sector collaboration is a must. However, if designers and engineers cannot understand each others’ field, it is very difficult to communicate complex ideas.
In order to address this difficulty, a device which can connect designers and engineers, and serve as a translator for their ideas is needed.

Some examples of such a device are “Gainer”, “Funnel” and “Arduino”, which have been developed as toolkits for a physical computing using a PC.

These toolkits are a small-size computer designed to be extension equipment for modern computer input and output systems. They are also designed to be easy to understand for designers who are not engineers. On the bounty of highly-modularized input and output systems, these toolkits are widely favoured among artists, designers and engineers as a tool for realising their physical computing ideas.

However, in recent years, it seems that the meaning of "computer in our lives is shifting from desktops and notebooks to smartphones and tablets. With that in mind, we developed Konashi as a physical computing toolkit tailored for smartphones and tablets. Anybody can take advantage of Konashi by using our open-source library "konashi-ios-sdk" and development environment for smartphone and tablet apps.


## Specifications

### Supported devices

<table>
<thead>
<tr>
<th>Series Name</th>
<th>Model</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>iPhone 4S, iPhone 5, iPhone 5S, iPhone 5C</td>
<td>iOS7.1 ～</td>
</tr>
<tr>
<td>iPad</td>
<td>iPad Air, iPad mini, iPad(4th Generation/ November 2012 Model), iPad(3rd Generation /March 2012 Model)</td>
<td>iOS7.1 ～</td>
</tr>
<tr>
<td>iPod touch</td>
<td>iPod touch(5th Generation)</td>
<td>iOS7.1 ～</td>
</tr>
</tbody>
</table>

Please note: Not compatible with 4th generation iPod touch

## Appearance
Top view

Bottom view

Layout

Schematic
konashi is equipped with 8 digital I/O pins. With digital I/O pins, it is possible to input and output in two states, high and low.

In konashi's case, the standard voltage for the digital I/O is 3 volts, so it can input and output in either 3 or 0 volts.

Furthermore, in their initial state all of the digital I/O are set to input.

If the digital I/O pins are set to input, it is also possible to configure the internal pull-up resistors with `pinMode`, `pinModeAll` functions.
Analog

Konashi is equipped with 3 analog I/O pins. With analog I/O, voltage is inputted as microvolts, and it is possible to output a specified voltage (DAC).

The base voltage of konashi’s analog I/O is 1.3 volts, so it can input and output in microvolts from 0 to 1.3 volts.

Furthermore, the initial state of the analog I/O is ADC (input).

Please be aware that ADC and DAC cannot be used at the same time.

Also, for DAC, only one pin can be used at a time. If you set other pins to DAC, another pin will be reset to input.

PWM

PWM (Pulse Width Modulation) is a system style which controls the ON time (duty cycle) of a pin, and generates pulses by turning it ON and OFF. This is a commonly-used method to control motor rotation speed or LED light strength.

On konashi, it is possible to set all of the digital I/O (PIO) pins to PWM mode.

When using PWM on konashi, in addition to the modes pwmMode, pwmPeriod, pwmDuty, which decide the duty cycle or period with a function, there is a mode which can easily specify the brightness of an LED from 0 to 100% (using the function pwmLedDrive). Which mode to use can be specified with the function pwmMode.

Because konashi’s pulse width modulation is packaged with pwm software, if the duty cycle is modified multiple times in a short interval, konashi’s ble systems will process each change consecutively, so it may not always output the specified duty cycle accurately.

Therefore, please try not to access konashi with ble consecutively, or if the duty cycle does get out of sync, please target and control something which will not be damaged.

Also, the lowest PWM cycle rate it is possible to set is 1000 [us].
Communication

In order to enable serial communication with other devices, konashi is compatible with \( \text{I}^2\text{C} \) and UART.

\( \text{I}^2\text{C} \)

The signal types used with \( \text{I}^2\text{C} \) Serial Data (SDA) and Serial Clock (SDL) only.

This signal standard was developed as a simple bus system used for controlling electronics, and the standard’s specifics can be found on the company NPX Semiconductor’s website. (Please refer below for references.)

Reference: \( \text{I}^2\text{C} \) Bus Specification Documents Version 2.1 (NPX Semiconductors) (PDF File/780KB)

konashi operates as an \( \text{I}^2\text{C} \) master, and carries out communication with connected (slave) peripherals.

The two signal waves SDA and SDL are compatible with POI6(SDA) and POI7(SCL), respectively. They are connected as outlined in the above diagram, and programmed according to a device’s signal protocol, so can communicate with LCDs and sensors which are compatible with \( \text{I}^2\text{C} \).

Please note, be sure to use pull-up resistors with SDA and SDL.

By using the \( \text{I}^2\text{C} \) functions in konashi’s library, it is possible to efficiently program \( \text{I}^2\text{C} \) communication.

\( \text{UART} \)

UART is a general term for a function which carries out serial communication through a start-stop synchronisation
system, and by making use of this it is possible to carry out serial communication with konashi, conforming to RS-232(ANSI/TIA/EIA-232-F-1997).

Konashi uses two signal waves; one for sending data (TX) and one for receiving data (RX), and carries out serial communication with UART. It is connected as shown in the above diagram, and with UART it is possible to easily communicate with PCs and other peripherals.

The voltage of the signal is 3V.

To see which signal speeds it is possible to set, please look at Constants / UART.

Bluetooth Low Energy

Bluetooth Low Energy (here referred to as BLE), is a low-energy consumption version of Bluetooth as decided by Bluetooth SIG, and it is equipped in iPhones, iPads, and the latest MacBooks.

Konashi uses BLE to communicate with iPhones and iPads.

As konashi is designed to be simple for first-time users, the CPU does not often enter sleep mode and such, thus it is not designed for low energy consumption. Please note that it uses more energy than most BLE devices.

Following is an outline of the UUIDs for konashi’s service and characteristics.

Services

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHI_SERVICE_UUID</td>
<td>0xFF00</td>
</tr>
</tbody>
</table>

Characteristics

PIO

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHIPIO_SETTING_UUID</td>
<td>0x3000</td>
</tr>
<tr>
<td>KONASHIPIO_PULLUP_UUID</td>
<td>0x3001</td>
</tr>
<tr>
<td>KONASHIPIO_OUTPUT_UUID</td>
<td>0x3002</td>
</tr>
<tr>
<td>KONASHIPIO_INPUT_NOTIFICATION_UUID</td>
<td>0x3003</td>
</tr>
</tbody>
</table>

PWM

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHI_PWM_CONFIG_UUID</td>
<td>0x3004</td>
</tr>
<tr>
<td>Name</td>
<td>UUID</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>KONASHI_PWM_PARAM_UUID</td>
<td>0x3005</td>
</tr>
<tr>
<td>KONASHI_PWM_DUTY_UUID</td>
<td>0x3006</td>
</tr>
</tbody>
</table>

**Analog**

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHI_ANALOG_DRIVE_UUID</td>
<td>0x3007</td>
</tr>
<tr>
<td>KONASHI_ANALOG_READ0_UUID</td>
<td>0x3008</td>
</tr>
<tr>
<td>KONASHI_ANALOG_READ1_UUID</td>
<td>0x3009</td>
</tr>
<tr>
<td>KONASHI_ANALOG_READ2_UUID</td>
<td>0x300A</td>
</tr>
</tbody>
</table>

**I2C**

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHI_I2C_CONFIG_UUID</td>
<td>0x300B</td>
</tr>
<tr>
<td>KONASHI_I2C_START_STOP_UUID</td>
<td>0x300C</td>
</tr>
<tr>
<td>KONASHI_I2C_WRITE_UUID</td>
<td>0x300D</td>
</tr>
<tr>
<td>KONASHI_I2C_READ_PARAM_UUID</td>
<td>0x300E</td>
</tr>
<tr>
<td>KONASHI_I2C_READ_UUID</td>
<td>0x300F</td>
</tr>
</tbody>
</table>

**UART**

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHI_UART_CONFIG_UUID</td>
<td>0x3010</td>
</tr>
<tr>
<td>KONASHI_UART_BAUDRATE_UUID</td>
<td>0x3011</td>
</tr>
<tr>
<td>KONASHI_UART_TX_UUID</td>
<td>0x3012</td>
</tr>
<tr>
<td>KONASHI_UART_RX_NOTIFICATION_UUID</td>
<td>0x3013</td>
</tr>
</tbody>
</table>

**Hardware control**

<table>
<thead>
<tr>
<th>Name</th>
<th>UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHI_HARDWARE_RESET_UUID</td>
<td>0x3014</td>
</tr>
<tr>
<td>KONASHI_HARDWARE_LOW_BAT_NOTIFICATION_UUID</td>
<td>0x3015</td>
</tr>
</tbody>
</table>
## API Reference

### Constants

#### Pin name

<table>
<thead>
<tr>
<th>Pin name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIO0</td>
<td>0</td>
<td>PIO pin 0</td>
</tr>
<tr>
<td>PIO1</td>
<td>1</td>
<td>PIO pin 1</td>
</tr>
<tr>
<td>PIO2</td>
<td>2</td>
<td>PIO pin 2</td>
</tr>
<tr>
<td>PIO3</td>
<td>3</td>
<td>PIO pin 3</td>
</tr>
<tr>
<td>PIO4</td>
<td>4</td>
<td>PIO pin 4</td>
</tr>
<tr>
<td>PIO5</td>
<td>5</td>
<td>PIO pin 5</td>
</tr>
<tr>
<td>PIO6</td>
<td>6</td>
<td>PIO pin 6</td>
</tr>
<tr>
<td>PIO7</td>
<td>7</td>
<td>PIO pin 7</td>
</tr>
<tr>
<td>S1</td>
<td>0</td>
<td>Tact switch (When the jumper is short-circuited, connected to PIO0)</td>
</tr>
<tr>
<td>LED2</td>
<td>1</td>
<td>Red LED (When the jumper is short-circuited, connected to PIO1)</td>
</tr>
<tr>
<td>LED3</td>
<td>2</td>
<td>Red LED (When the jumper is short-circuited, connected to PIO2)</td>
</tr>
<tr>
<td>LED4</td>
<td>3</td>
<td>Red LED (When the jumper is short-circuited, connected to PIO3)</td>
</tr>
<tr>
<td>LED5</td>
<td>4</td>
<td>Red LED (When the jumper is short-circuited, connected to PIO4)</td>
</tr>
<tr>
<td>AIO0</td>
<td>0</td>
<td>AIO pin 0</td>
</tr>
<tr>
<td>AIO1</td>
<td>1</td>
<td>AIO pin 1</td>
</tr>
<tr>
<td>AIO2</td>
<td>2</td>
<td>AIO pin 2</td>
</tr>
<tr>
<td>I2C_SDA</td>
<td>6</td>
<td>I²C SDA pin (PIO pin 6)</td>
</tr>
<tr>
<td>I2C_SCL</td>
<td>7</td>
<td>I²C SCL pin (PIO pin 7)</td>
</tr>
</tbody>
</table>

## PIO

<table>
<thead>
<tr>
<th>Mode</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGH</td>
<td>1</td>
<td>Changes the pin’s output to HIGH (3V)</td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>Changes the pin’s output to LOW(0V)</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>True</td>
<td>1</td>
<td>Changes the pin’s output to HIGH(3V) (Same as the above “HIGH”)</td>
</tr>
<tr>
<td>False</td>
<td>0</td>
<td>Changes the pin’s output to LOW(0V) (Same as the above “LOW”)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>1</th>
<th>Sets the pin’s input/output to output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>0</td>
<td>Set the pin’s input/output to input</td>
</tr>
<tr>
<td>Pullup</td>
<td>1</td>
<td>Set the pin’s pull-up setting to ON</td>
</tr>
<tr>
<td>No Pulls</td>
<td>0</td>
<td>Set the pin’s pull-up setting to OFF</td>
</tr>
</tbody>
</table>

**AIO**

<table>
<thead>
<tr>
<th>Konashi_Analog_Reference</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHI_ANALOG_REFERENCE</td>
<td>1300</td>
<td>Reference voltage for analog input and output 1300mV</td>
</tr>
</tbody>
</table>

**PWM**

<table>
<thead>
<tr>
<th>Konashi_PWM_Disable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHI_PWM_DISABLE</td>
<td>0</td>
<td>Do not use the specified PIO as PWM (Use as digital I/O)</td>
</tr>
<tr>
<td>KONASHI_PWM_ENABLE</td>
<td>1</td>
<td>Use the specified PIO as PWM</td>
</tr>
<tr>
<td>KONASHI_PWM_ENABLE_LED_MODE</td>
<td>2</td>
<td>Use the specified PIO as a PWM in LED mode</td>
</tr>
<tr>
<td>KONASHI_PWM_LED_PERIOD</td>
<td>10000</td>
<td>When in LED mode the PWM cycle is 10ms</td>
</tr>
</tbody>
</table>

**UART**

<table>
<thead>
<tr>
<th>Konashi_UART_Disable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHI_UART_DISABLE</td>
<td>0</td>
<td>Disable UART</td>
</tr>
<tr>
<td>KONASHI_UART_ENABLE</td>
<td>1</td>
<td>Enable UART</td>
</tr>
<tr>
<td>KONASHI_UART_RATE_2K4</td>
<td>0x000a</td>
<td>2400bps</td>
</tr>
<tr>
<td>KONASHI_UART_RATE_9K6</td>
<td>0x0028</td>
<td>9600bps</td>
</tr>
</tbody>
</table>
### I²C

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHI_I2C_DATA_MAX_LENGTH</td>
<td>19</td>
<td>The largest amount of bytes I²C can receive at one time</td>
</tr>
<tr>
<td>KONASHI_I2C_DISABLE</td>
<td>0</td>
<td>Disable I²C</td>
</tr>
<tr>
<td>KONASHI_I2C_ENABLE</td>
<td>1</td>
<td>Enable I²C to be enabled (Default mode: 100kbps)</td>
</tr>
<tr>
<td>KONASHI_I2C_ENABLE_100K</td>
<td>1</td>
<td>Enable I²C in 100kbps mode</td>
</tr>
<tr>
<td>KONASHI_I2C_ENABLE_400K</td>
<td>2</td>
<td>Enable I²C in 400kbps mode</td>
</tr>
<tr>
<td>KONASHI_I2C_STOP_CONDITION</td>
<td>0</td>
<td>Stop Condition</td>
</tr>
<tr>
<td>KONASHI_I2C_START_CONDITION</td>
<td>1</td>
<td>Start Condition</td>
</tr>
<tr>
<td>KONASHI_I2C_RESTART_CONDITION</td>
<td>2</td>
<td>Restart Condition</td>
</tr>
</tbody>
</table>

### Function return

<table>
<thead>
<tr>
<th>Function</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHI_SUCCESS</td>
<td>0</td>
<td>When successful</td>
</tr>
<tr>
<td>KONASHI_FAILURE</td>
<td>1</td>
<td>When failure</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KONASHI_EVENT_CONNECTED</td>
<td>When konashi is connected (The connection has not finished)</td>
</tr>
<tr>
<td>KONASHI_EVENT_READY</td>
<td>When the connection with konashi is finished</td>
</tr>
<tr>
<td>KONASHI_EVENT_UPDATEPIO_INPUT</td>
<td>When PIO’s input status has changed</td>
</tr>
<tr>
<td>KONASHI_EVENT_UPDATEANALOGVALUE</td>
<td>When the voltage for one of the AIO pins is acquired</td>
</tr>
<tr>
<td>KONASHI_EVENT_UPDATEANALOGVALUE_AIO0</td>
<td>When the voltage for AIO0 is acquired</td>
</tr>
<tr>
<td>KONASHI_EVENT_UPDATEANALOGVALUE_AIO1</td>
<td>When the voltage for AIO1 is acquired</td>
</tr>
<tr>
<td>KONASHI_EVENT_UPDATEANALOGVALUE_AIO2</td>
<td>When the voltage for AIO2 is acquired</td>
</tr>
<tr>
<td>KONASHI_EVENT_I2C_READ_COMPLETE</td>
<td>When data is received from I²C</td>
</tr>
<tr>
<td>KONASHI_EVENT_UART_RX_COMPLETE</td>
<td>When data is received from UART’s Rx</td>
</tr>
</tbody>
</table>
KONASHI_EVENT_UPDATE_BATTERY_LEVEL | When konashi’s battery level is acquired

KONASHI_EVENT_UPDATE_SIGNAL_STRENGTH | When konashi’s signal strength is acquired

---

**Base**

**initialize**

**Description**

Initialising konashi.

Before using konashi, be sure to initialize viewDidLoad, and anything else which appears first on ViewController.

**Syntax**

```
[Konashi initialize];
```

**Parameters**

None

**Returns**

If successful: KONASHISUCCESS (0), If a failure: KONASHIFailure (-1)

---

**find**

**Description**

Find a konashi near to your iPhone

After executing this function, a list of konashi around your phone will appear. If you click on one of them, you will connect to it automatically. After that, the events KONASHI_EVENT_CONNECTED and KONASHI_EVENT_READY will launch, so please arrange to catch these events on addObserver beforehand.

**Syntax**

```
[Konashi find];
```

**Parameters**
Returns

If successful: KONASHISUCCESS (0), If a failure: KONASHIFailure (-1)

disconnect

Description

Disconnect konashi.

Notice: In operating systems older than iOS6.1, there is a bug in the Core Bluetooth API, so this function will not work properly. Please use iOS6.1 or later.

Syntax

```
[Konashi disconnect];
```

Parameters

None

Returns

If successful: KONASHISUCCESS (0), If unsuccessful: KONASHIFailure (-1)

isConnected

Description

Return if the iOS device is connected to konashi.

When the event KONASHI_EVENT_CONNECTED launches it becomes TRUE. Before that it is FALSE.

Syntax

```
[Konashi isConnected];
```

Parameters

None
Events

addObserver

Description

You can catch events related to konashi.

Because konashi and the iPhone are connected by BLE, konashi’s status and the status of pins are acquired asynchronously.

For example, to acquire the voltage of an AIO pin, send a request to konashi with `analogReadRequest`, and after receiving the acquisition complete event `KONASHI_EVENT_UPDATE_ANALOG_VALUE`, it becomes possible to reference the AIO pin’s voltage.

To see what kind of events there are please reference Constants / Events.

Syntax

```objective-c
[Konashi addObserver:(id)notificationObserver selector:(SEL)notificationSelector name:(NSString*)notificationName];
```

Parameters

<table>
<thead>
<tr>
<th>notificationObserver</th>
<th>id</th>
<th>Specify the observer</th>
</tr>
</thead>
<tbody>
<tr>
<td>selector</td>
<td>SEL</td>
<td>Specify a function to call an event launch</td>
</tr>
<tr>
<td>name</td>
<td>NSString*</td>
<td>Specify an event name. For details, please reference Constants - Events</td>
</tr>
</tbody>
</table>

Returns

If successful: `KONASHI_SUCCESS` (0), If unsuccessful: `KONASHI_FAILURE` (-1)

Example

```objective-c
-(void)viewDidLoad
{
    [super viewDidLoad];
    [Konashi initialize];
}
```
removeObserver

Description
Cancel konashi event observation carried out by `addObserver`.

Syntax

```objc
[Konashi removeObserver:(id)notificationObserver];
```

Parameters

- `notificationObserver` id Specify the observer

Returns

If successful: `KONASHI_SUCCESS` (0), if unsuccessful: `KONASHI_FAILURE` (-1)

Example

```objc
- (void)viewDidLoad
{
    [super viewDidLoad];

    [Konashi initialize];
    [Konashi addObserver:self selector:@selector(konashiConnected)
name:KONASHI_EVENT_CONNECTED];
}

- (void)konashiConnected
{
    NSLog(@"CONNECTED");
    [Konashi removeObserver:self];
}
```
Digital I/O (PIO)

pinMode

Description
Specify whether to use a PIO pin as input or output. For details please look at Core functions / Digital.

Syntax
```
[Konashi pinMode:(int)pin mode:(int)mode];
```

Parameters

<table>
<thead>
<tr>
<th>pin</th>
<th>int</th>
<th>Specified PIO pin name. To see what pin names are possible please look at Constants / Pin name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>int</td>
<td>Specified mode for that pin. You can specify whether it is INPUT or OUTPUT. For details please look at Constants / PIO.</td>
</tr>
</tbody>
</table>

Returns
If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

Example
Set LED2 as OUTPUT
```
[Konashi pinMode:LED2 mode:OUTPUT];
```

pinModeAll

Description
Please specify whether to use the PIO pin for input or output. For details please see Core functions / Digital.

Syntax
```
[Konashi pinModeAll:(int)mode];
```

Parameters
mode int The settings of all 8 pins from PIO0 to PIO7. Please specify from 0x00 to 0xFF, with 1 as OUTPUT, and 0 as INPUT.

Returns
If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

Example
Set all PIO pins as OUTPUT

[Konashi pinModeAll:0xFF];

pinPullup

Description
Specify whether to set pull-up for PIO pins.

In their initial state, no pull-up is applied to the PIO pins(NO_PULLS). Please look at Core Functions / Digital for details.

Syntax

[Konashi pinPullup:(int)pin mode:(int)mode];

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>int</td>
<td>The name of a specified PIO pin. Please see Constants / Pin name for possible pin names.</td>
</tr>
<tr>
<td>mode</td>
<td>int</td>
<td>Specify whether to set pull-up for the pin. PULLUP or NO_PULLS can be specified. For details, please look at Constants / PIO.</td>
</tr>
</tbody>
</table>

Returns
If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

Example
Set pull-up for PIO7

[Konashi pinMode:PIO7 mode:PULLUP];
**pinPullupAll**

**Description**
Specify whether to implement a pull-up resistor for PIO pins. For details, please see Core Functions / Digital.

**Syntax**

```konashi
[Konashi pinPullupAll:(int)mode];
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>int</td>
<td>Name of the specified PIO pin. For possible names please see Constants / Pin name.</td>
</tr>
<tr>
<td>mode</td>
<td>int</td>
<td>Settings for the pull-up resistors of all pins from PIO0 through to PIO7. Please specify from 0x00 to 0xFF.</td>
</tr>
</tbody>
</table>

**Returns**
If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

**Example**
Set pull-up for All PIO pins

```konashi
[Konashi pinPullupAll:0xFF];
```

**digitalRead**

**Description**
Acquire the input condition of a specific PIO pin.

Before acquiring the input condition of a pin, be sure to set the pin’s mode to input with pinMode, pinModeAll. If the pin is in output mode the correct input condition cannot be acquired.

For details, please see Core functions / Digital.

**Syntax**

```konashi
[Konashi digitalRead:(int)pin];
```

**Parameters**
pin int Name of the PIO pin. For possible names, please look at Constants / Pin name.

Returns

HIGH or LOW

Example

Acquire the input condition of S1

[Konashi digitalWrite:S1];

digitalReadAll

Description

Acquire the conditions of all PIO pins. For details, please see Core functions / Digital.

Syntax

[Konashi digitalWriteAll];

Parameters

None.

Returns

Input information for the 8 PIO pins (0x00 to 0xFF)

Example

Acquire the input condition for all of the PIO pins.

[Konashi digitalWriteAll];

digitalWrite

Description

Set the output condition for a specific PIO pin.
Syntax

```c
[Konashi digitalWrite:(int)pin value:(int)value];
```

Parameters

- **pin** int Name of the PIO pin. For possible pin names please see *Constants / Pin name*.
- **value** int The output condition for the assigned PIO. `HIGH` or `LOW` can be specified. For details please see *Constants / PIO*.

Returns

- If successful: `KONASHI_SUCCESS`, If unsuccessful: `KONASHI_FAILURE`

Example

Set LED2 to HIGH

```c
[Konashi digitalWrite:LED2 value:HIGH];
```

digitalWriteAll

Description

Set the output conditions for a specific PIO pin. For details, please look at *Core functions / Digital*.

Syntax

```c
[Konashi digitalWriteAll:(int)value];
```

Parameters

- **value** int A value set from output PIO0 to PIO7. It is possible to set it between 0x00 and 0xFF.

Returns

- If successful: `KONASHI_SUCCESS`, If unsuccessful: `KONASHI_FAILURE`

Example

Set all of the PIO pin on HIGH.
Analog I/O (AIO)

analogReference

Description
Return the analog output to the base voltage

Syntax
[Konashi analogReference];

Parameters
None

Returns
KONASHI_ANALOG_REFERENCE, return to 1300(mV).

analogReadRequest

Description
Send a request to konashi to acquire the voltage of a specified AIO pin.

This is a function to send a request to konashi, so to acquire the value practically then after checking the voltage acquisition completion event, defined in KONASHI_EVENT_UPDATE_ANALOG_VALUE or Constants / Events, in addObserver, acquire the value with analogRead.

For features related to analog functions, please see Core functions / Analog.

Syntax
[Konashi analogReadRequest:(int)pin];

Parameters
Name of an AIO pin. The possible pin names are \texttt{AIO0}, \texttt{AIO1} and \texttt{AIO2}. For details please see \texttt{Constants / Pin name}.

### Returns

If successful: \texttt{KONASHI_SUCCESS}, If unsuccessful: \texttt{KONASHI_FAILURE}

### Example

Acquire the input voltage of AIO0.

```objective-c
-(void)viewDidLoad
{
    [super viewDidLoad];
    [Konashi initialize];
    [Konashi addObserver:self selector:@selector(konashiReady) name:KONASHI_EVENT_READY];
    [Konashi addObserver:self selector:@selector(readAio0) name:KONASHI_EVENT_UPDATE_ANALOG_VALUE_AIO0];
}
// Tap action for the search for konashi button
-(IBAction)findKonashi:(id)sender {
    [Konashi find];
}
// Tap action for button to acquire voltage for AIO0
-(IBAction)requestReadAio0:(id)sender {
    [Konashi analogReadRequest:AIO0];
}
-(void)readAio0
{
    NSLog(@"READ_AIO0: \(d)", [Konashi analogRead:AIO0]);
}
```

### analogRead

**Description**

Acquire the voltage for a specified AIO pin. The value which can be acquired with this function is the voltage acquired with \texttt{analogReadRequest} previously.

If you would like to acquire the voltage of konashi’s AIO pins, first, send an acquisition request to konashi with \texttt{analogReadRequest}, then after checking the voltage acquisition complete event defined in
KONASHI\EVENT\UPDATE\ANALOG\VALUE or Constants / Events with addObserver, then you can obtain the value with this function.

For functions related to analog features, please see Core functions / Analog.

**Syntax**

```plaintext
[Konashi analogRead:(int)pin];
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>Name of an AIO pin. The possible pin names are AIO0, AIO1 and AIO2. For details please see Constants / Pin name.</td>
</tr>
</tbody>
</table>

**Returns**

A mV unit value from 0 to KONASHI\ANALOG\REFERENCE will be returned.

**Example**

Obtain input voltage for AIO0

```swift
- (void)viewDidLoad
{
    [super viewDidLoad];

    [Konashi initialize];

    [Konashi addObserver:self selector:@selector(konashiReady) name:KONASHI_EVENT_READY];
    [Konashi addObserver:self selector:@selector(readAio0) name:KONASHI_EVENT_UPDATE_ANALOG_VALUE_AIO0];
}

// Tap action for search for konashi button
- (IBAction)findKonashi:(id)sender {
    [Konashi find];
}

// Tap action for button to obtain voltage for AIO0
- (IBAction)requestReadAio0:(id)sender {
    [Konashi analogReadRequest:AIO0];
}

- (void)readAio0
{
    NSLog(@"READ_AIO0: \%d", [Konashi analogRead:AIO0]);
}
```
**analogWrite**

**Description**

Output an unspecified voltage with a specified AIO pin.

The highest voltage which can be set is KONASHI_ANALOGREFERENCE.

For analogue features, please see Core functions / Analog.

**Syntax**

```
[Konashi analogWrite:(int)pin milliVolt:(int)milliVolt];
```

**Parameters**

- **pin** int  Name of an AIO pin. The possible names are AIO0, AIO1, and AIO2. For details please see Constants / Pin name.

- **milliVolt** int  Set the specified voltage in mV. It is possible to set it from 0 to `KONASHI_ANALOG_REFERENCE`.

**Returns**

If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

**Example**

Make AIO output 1.0V

```
[Konashi analogWrite:AIO0 milliVolt:1000];
```

**PWM**

**pwmMode**

**Description**

Set whether to use a specified PIO pin as PWM

Any PIO pin can be set to PWM mode.

When assigning KONASHI_PWM_ENABLE mode, please specify the cycle and ON period with pwmPeriod and pwmDuty beforehand.
For details on PWM, please see Core functions / PWM.

Syntax

```
[Konashi pwmMode:(int)pin mode:(int)mode];
```

Parameters

- **pin** int: The name of the pin set to PWM mode. It is possible to set from `PIO0` to `PIO7`.
- **mode** int: The set PWM mode. It is possible to specify KONASHI_PWM_DISABLE, KONASHI_PWM_ENABLE, or KONASHI_PWM_ENABLE_LED_MODE. For details please see Constants / PWM.

Returns

If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

Example

Set PWM with the LED cycle set to 10ms, and duty cycle to 5ms

```
[Konashi pwmMode:LED2 mode:KONASHI_PWM_ENABLE];
[Konashi pwmPeriod:LED2 period:10000];
[Konashi pwmDuty:LED2 duty:5000];
```

**pwPeriod**

Description

Set the PWM cycle of a specified pin.

Please specify the cycle units as microseconds (us).

For details on PWM please see Core functions / PWM.

Syntax

```
[Konashi pwmPeriod:(int)pin period:(unsigned int)period];
```

Parameters

- **pin** int: Name of a PIO pin. `PIO0` to `PIO7` can be set.
period  unsigned  int  Please specify the units as microseconds (us) in 32 bits. The largest possible is $2^{32}\text{us} = 71.5$ minutes.

**Returns**

If successful: `KONASHI_SUCCESS`, If unsuccessful: `KONASHI_FAILURE`

**Example**

Set LED2's period to 10ms

```
[Konashi pwmPeriod:LED2 period:10000];
```

**pwmDuty**

**Description**

Set the duty cycle of a specified pin (when on).

Please specify the units in microseconds (us).

For details on PWM please see [Core functions / PWM](#).

**Syntax**

```
[Konashi pwmDuty: (int) pin duty: (unsigned int) duty];
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>int</td>
<td>Name of a PIO pin. <code>PIO0</code> to <code>PIO7</code> can be set.</td>
</tr>
<tr>
<td>duty</td>
<td>unsigned int</td>
<td>Duty cycle. Please specify the units as microseconds (us) in 32 bits. The largest possible is $2^{32}\text{us}, 71.5$ minutes.</td>
</tr>
</tbody>
</table>

**Returns**

If successful: `KONASHI_SUCCESS`, If unsuccessful: `KONASHI_FAILURE`

**Example**

Set the duty cycle for LED2 to 5ms

```
[Konashi pwmDuty:LED2 duty:5000];
```
pwmLedDrive

Description

Set the brightness of the LED of a specified pin from 0 to 100%

When using the function `pwmLedDrive`, please specify KONASHI_PWMENABLELEDMODE with `pwmMode`.

For details on PWM please see Core functions / PWM.

Syntax

```
[Konashi pwmLedDrive:(int)pin dutyRatio:(int)ratio];
```

Parameters

<table>
<thead>
<tr>
<th>pin</th>
<th>int</th>
<th>Name of a PIO pin. It is possible to set from <code>PIO0</code> to <code>PIO7</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ratio</td>
<td>int</td>
<td>Brightness of the LED. Please set from 0 to 100.</td>
</tr>
</tbody>
</table>

Returns

If successful: `KONASHI_SUCCESS`, If unsuccessful: `KONASHI_FAILURE`

Example

Set the brightness of LED2 to 30%

```
[Konashi pwmMode:LED2 mode:KONASHI_PWM_ENABLE_LED_MODE];
[Konashi pwmLedDrive:LED2 dutyRatio:30];
```

UART

uartMode

Description

Enable or disable UART.

Before enabling, please set the Baud Rate with `uartBaudrate`.

For details on UART, please see Core functions / Communication - UART.

Syntax
[Konashi uartMode:(int)mode];

**Parameters**

| mode   | int   | The set UART mode. KONASHI_UART_DISABLE or KONASHI_UART_ENABLE can be set. |

**Returns**

If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

**Example**

Enable UART

[Konashi uartMode:KONASHI_UART_ENABLE];

---

**uartWrite**

**Description**

Send one byte of data with UART.

For details on UART, please see Core functions / Communication - UART.

**Syntax**

[Konashi uartWrite:(unsigned char)data];

**Parameters**

| data       | unsigned char | The data that will be sent. 1 byte. |

**Returns**

If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

**Example**

Send the character ‘A’ with UART

[Konashi uartWrite:'A'];
I2C

i2cMode

Description
Enable or Disable I²C.

Communication speed of I²C can be set to 100kbps or 400kbps, which is specified with the `mode` argument.

For details on I²C please see Core functions / Communication - I²C.

Syntax

```
[Konashi i2cMode:(int)mode];
```

Parameters

| mode  | int   | Set mode for I²C. KONASHI_I2C_DISABLE, KONASHI_I2C_ENABLE, KONASHI_I2C_ENABLE_100K, or KONASHI_I2C_ENABLE_400K can be set. KONASHI_I2C_ENABLE and KONASHI_I2C_ENABLE_100K are equivalent |

Returns

If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

Example

Enable 100kbps(Default) communication speed on I²C

```
[Konashi i2cMode:KONASHI_I2C_ENABLE];
```

i2cStartCondition

Description
Raise I²C's start condition.

Please enable I²C with i2cMode beforehand.

For details on I²C, please see Core functions / Communication - I²C.

Syntax
i2cStartCondition

Parameters
None

Returns
If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

i2cRestartCondition

Description
Raise I²C’s restart condition.

Please enable I²C with i2cMode beforehand.

For details on I²C, please see Core functions / Communication - I²C.

Syntax

[Konashi i2cRestartCondition];

Parameters
None

Returns
If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

i2cStopCondition

Description
Raise I²C’s stop condition.

Please enable I²C with i2cMode beforehand.

For details on I²C, please see Core functions / Communication - I²C.

Syntax

[Konashi i2cStopCondition];
i2cWrite

Description

Write data to a specified address with I²C.

Please enable I²C with i2cMode beforehand.

For details on I²C, please see Core functions / Communication - I²C.

Syntax

[Konashi i2cWrite:(int)length data:(unsigned char*)data address:(unsigned char)address];

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>int</td>
<td>Length of the data to be written (bytes). The longest possible,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KONASHI_I2C_DATA_MAX_LENGTH is 19bytes.</td>
</tr>
<tr>
<td>data</td>
<td>unsigned char*</td>
<td>The date to be written</td>
</tr>
<tr>
<td>address</td>
<td>unsigned char</td>
<td>The address to be written to</td>
</tr>
</tbody>
</table>

Returns

If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

Example

Write the data ‘A’ to the address 0x45

```c
unsigned char data[] = {'A'};
[Konashi i2cWrite:1 data:data address:0x45];
```
i2cReadRequest

Description

Make a request to read data from the address specified with I²C.

With this function, data cannot be obtained by only making a request. In order to gain the value, after checking `KONASHI_EVENT_I2C_READ_COMPLETE` with `addObserver`, obtain the value with `i2cRead`.

Syntax

```
[Konashi i2cReadRequest:(int)length address:(unsigned char)address];
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>int</td>
<td>Length of the data to be read. The max length, <code>KONASHI_I2C_DATA_MAX_LENGTH</code> (19)</td>
</tr>
<tr>
<td>address</td>
<td>unsigned char</td>
<td>Address to read the data from</td>
</tr>
</tbody>
</table>

Returns

If successful: KONASHISUCCESS, If unsuccessful: KONASHIFailure

i2cRead

Description

Konashi will obtain the data read by I²C. The value obtained with this function is of the time when previous `i2cReadRequest` was issued.

Syntax

```
[Konashi i2cRead:(int)length data:(unsigned char*)data];
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>int</td>
<td>The length of the data to be read. The longest possible, <code>KONASHI_I2C_DATA_MAX_LENGTH</code> (19)</td>
</tr>
<tr>
<td>data</td>
<td>unsigned char</td>
<td>Memory pointer for the data read will be stored</td>
</tr>
</tbody>
</table>
Returns
If successful: **KONASHI_SUCCESS**, If unsuccessful: **KONASHI_FAILURE**

**Hardware Control**

**reset**

**Description**
Restart konashi.

If you restart konashi, the BLE connection will be automatically closed.

**Syntax**

```
[Konashi reset];
```

**Parameters**
None

**Returns**
If successful: **KONASHI_SUCCESS**, If unsuccessful: **KONASHI_FAILURE**

**Example**
Restart konashi

```
[Konashi reset];
```

**batteryLevelReadRequest**

**Description**
Send a request to konashi to obtain the remaining battery level.

This function sends a request to konashi, so in order to acquire the value, after checking KONASHI_EVENTUPDATEBATTERYLEVEL (Battery level acquisition complete event) in `addObserver`, you can obtain the value with `batteryLevelRead`. 
Syntax

[Konashi batteryLevelReadRequest];

Parameters

None

Returns

If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

Example

Obtain konashi’s remaining battery level.

```swift
- (void)viewDidLoad
{
    [super viewDidLoad];

    [Konashi initialize];

    [Konashi addObserver:self selector:@selector(konashiReady) name:KONASHI_EVENT_READY];
    [Konashi addObserver:self selector:@selector(battery) name:KONASHI_EVENT_UPDATE_BATTERY_LEVEL];
}

// Tap function for button to search for konashi
- (IBAction)findKonashi:(id)sender {
    [Konashi find];
}

// Tap function for button to check battery level
- (IBAction)batteryLevelReadRequest:(id)sender {
    [Konashi batteryLevelReadRequest];
}

- (void)battery
{
    NSLog(@"READ_BATTERY: %d", [Konashi batteryLevelRead]);
}
```

batteryLevelRead

Description
konashi’s battery level will be obtained. The value which can be obtained with this function is the value obtained before with `batteryLevelReadRequest`.

If you wish to obtain konashi’s current battery level, first, send a request to konashi with `batteryLevelReadRequest`, and after checking KONASHI_EVENT_UPDATE_BATTERY_LEVEL with `addObserver`, you can obtain the value with this function.

**Syntax**

```
[Konashi batteryLevelRead];
```

**Parameters**

None

**Returns**

The remaining battery power level is returned as a percentage from 0 to 100.

**Example**

Acquire remaining battery power.

```swift
- (void)viewDidLoad
{
    [super viewDidLoad];

    [Konashi initialize];

    [Konashi addObserver:self selector:@selector(konashiReady) name:KONASHI_EVENT_READY];
    [Konashi addObserver:self selector:@selector(battery) name:KONASHI_EVENT_UPDATE_BATTERY_LEVEL];
}

// Search for konashi button tap action
- (IBAction)findKonashi:(id)sender {
    [Konashi find];
}

// Acquire remaining battery power button tap action
- (IBAction)batteryLevelReadRequest:(id)sender {
    [Konashi batteryLevelReadRequest];
}

- (void) battery
{
    NSLog(@"READ_BATTERY: %d", [Konashi batteryLevelRead]);
}
```
signalStrengthReadRequest

Description

Make a request to obtain konashi’s signal strength.

With this function, data cannot be obtained only by making a request. In order to obtain the value, after checking KONASHIEVENTUPDATESSIGNALSTRENGTH (Signal strength acquisition complete event) with addObserver, the value can be obtained with signalStrengthRead.

Syntax

[Konashi signalStrengthReadRequest];

Parameters

None

Returns

If successful: KONASHI_SUCCESS, If unsuccessful: KONASHI_FAILURE

Example

Obtain konashi’s signal strength

- (void)viewDidLoad
  {
    [super viewDidLoad];

    [Konashi initialize];

    [Konashi addObserver:self selector:@selector(konashiReady) name:KONASHI_EVENT_READY];
    [Konashi addObserver:self selector:@selector(strength) KONASHI_EVENT_UPDATE_SIGNAL_STRENGTH];
  }

  // Search for konashi button tap action
  - (IBAction)findKonashi:(id)sender {
    [Konashi find];
  }

  // Aquire signal strenght button tap action
  - (IBAction)signalStrengthReadRequest:(id)sender {
    [Konashi signalStrengthReadRequest];
  }

  - (void) strength
signalStrengthRead

Description

Obtain konashi’s signal strength. The value which can be obtained with this function is the strength (db) of the time when the previous signalStrengthReadRequest was issued.

If you wish to obtain konashi’s current signal strength, first, send a request to konashi with signalStrengthReadRequest, and after checking KONASHI_EVENT_UPDATESIGNALSTRENGTH in addObserver, the value can be obtained with this function.

Syntax

[Konashi signalStrengthRead];

Parameters

None

Returns

Return signal strength in db

Example

Acquire konashi’s signal strength

- (void)viewDidLoad
{   [super viewDidLoad];

    [Konashi initialize];

    [Konashi addObserver:self selector:@selector(konashiReady) name:KONASHI_EVENT_READY];
    [Konashi addObserver:self selector:@selector(strength) KONASHI_EVENT_UPDATE_SIGNAL_STRENGTH];
}

// Search for konashi button tap action
- (IBAction)findKonashi:(id)sender {
    [Konashi find];
}
// Acquire signal strength button tap action
- (IBAction)signalStrengthReadRequest:(id)sender {
    [Konashi signalStrengthReadRequest];
}

- (void) strength
{
    NSLog(@"READ_STRENGTH: \%d", [Konashi signalStrengthRead]);
}