# Firmata library reference

Generated by Doxygen 1.8.16

Thu Jan 2 2020 03:33:14

# Chapter 1

# Firmata

Firmata is a protocol for communicating with microcontrollers from software on a host computer. The `protocol` can be implemented in firmware on any microcontroller architecture as well as software on any host computer software package. The Arduino repository described here is a Firmata library for Arduino and Arduino-compatible devices. If you would like to contribute to Firmata, please see the `Contributing` section below.

## 1.1 Contents

### 1.1.1 Usage

There are two main models of usage of Firmata. In one model, the author of the Arduino sketch uses the various methods provided by the Firmata library to selectively send and receive data between the Arduino device and the software running on the host computer. For example, a user can send analog data to the host using `Firmata.sendAnalog(analogPin, analogRead(analogPin))` or send data packed in a string using `Firmata.sendString(stringToSend)`. See File -> Examples -> Firmata -> AnalogFirmata & EchoString respectively for examples. Browse the API documentation `here`.

The second and more common model is to load a general purpose sketch called StandardFirmata (or one of the variants such as StandardFirmataPlus or StandardFirmataEthernet depending on your needs) on the Arduino board and then use the host computer exclusively to interact with the Arduino board. StandardFirmata is located in the Arduino IDE in File -> Examples -> Firmata.

### 1.1.2 Firmata Client Libraries

Most of the time you will be interacting with Arduino with a client library on the host computers. Several Firmata client libraries have been implemented in a variety of popular programming languages:

- processing
  - https://github.com/firmata/processing
  - http://funnel.cc
- python
  - https://github.com/MrYsLab/pymata-aio
  - https://github.com/MrYsLab/PyMata
  - https://github.com/tino/pyFirmata
  - https://github.com/lupeke/python-firmata
  - https://github.com/firmata/pyduino
- perl
  - https://github.com/ntruchsess/perl-firmata
  - https://github.com/rcaputo/rx-firmata
- ruby
  - https://github.com/hardbap/firmata
  - https://github.com/PlasticLizard/rufinol
  - http://funnel.cc
- clojure
  - https://github.com/nakkaya/clodiuno
  - https://github.com/peterschwarz/clj-firmata
- javascript
  - https://github.com/firmata/firmata.js
  - https://github.com/rwldrn/johnny-five
  - http://breakoutjs.com
- java
  - https://github.com/kurbatov/firmata4j
  - https://github.com/4ntoine/Firmata
  - https://github.com/reapzor/FiloFirmata
- .NET
  - https://github.com/SolidSoils/Arduino
  - http://www.acraigie.com/programming/firmatavb/default.html
- Flash/AS3
  - http://funnel.cc
  - http://code.google.com/p/as3glue/
- Pharo
  - https://github.com/pharo-iot/Firmata

- PHP
  - [ https://github.com/ThomasWeinert/carica-firmata]()
  - https://github.com/oasynnoum/phpmake_firmata
- Haskell
  - http://hackage.haskell.org/package/hArduino
- iOS
  - https://github.com/jacobrosenthal/iosfirmata
- Dart
  - https://github.com/nfrancois/firmata
- Max/MSP
  - http://www.maxuino.org/
- Elixir
  - https://github.com/kfatehi/firmata
- Modelica
  - https://www.wolfram.com/system-modeler/libraries/model-plug/
- Go
  - https://github.com/kraman/go-firmata
- vvvv
  - https://vvvv.org/blog/arduino-second-service
- openFrameworks
  - http://openframeworks.cc/documentation/communication/ofArduino/
- Rust
  - https://github.com/zankich/rust-firmata

Note: The above libraries may support various versions of the Firmata protocol and therefore may not support all features of the latest Firmata spec nor all Arduino and Arduino-compatible boards. Refer to the respective projects for details.

### 1.1.3 Updating Firmata in the Arduino IDE - Arduino 1.6.4 and higher

If you want to update to the latest stable version:

1. Open the Arduino IDE and navigate to: `Sketch > Include Library > Manage Libraries`
2. Filter by "Firmata" and click on the "Firmata by Firmata Developers" item in the list of results.
3. Click the `Select version` dropdown and select the most recent version (note you can also install previous versions)
4. Click `Install`.

#### 1.1.3.1 Cloning Firmata

If you are contributing to Firmata or otherwise need a version newer than the latest tagged release, you can clone Firmata directly to your Arduino/libraries/ directory (where 3rd party libraries are installed). This only works for Arduino 1.6.4 and higher, for older versions you need to clone into the Arduino application directory (see section below titled "Using the Source code rather than release archive"). Be sure to change the name to Firmata as follows:

```
$ git clone git@github.com:firmata/arduino.git  /Documents/Arduino/libraries/Firmata
```

*Update path above if you're using Windows or Linux or changed the default Arduino directory on OS X*

### 1.1.4 Updating Firmata in the Arduino IDE - older versions (<= 1.6.3 or 1.0.x)

Download the latest `release` (for Arduino 1.0.x or Arduino 1.5.6 or higher) and replace the existing Firmata folder in your Arduino application. See the instructions below for your platform.

*Note that Arduino 1.5.0 - 1.5.5 are not supported. Please use Arduino 1.5.6 or higher (or Arduino 1.0.5 or 1.0.6).*

#### 1.1.4.1 Mac OSX:

The Firmata library is contained within the Arduino package.

1. Navigate to the Arduino application
2. Right click on the application icon and select `Show Package Contents`
3. Navigate to: `/Contents/Resources/Java/libraries/` and replace the existing `Firmata` folder with latest `Firmata release` (note there is a different download for Arduino 1.0.x vs 1.6.x)
4. Restart the Arduino application and the latest version of Firmata will be available.

*If you are using the Java 7 version of Arduino 1.5.7 or higher, the file path will differ slightly: `Contents/↩ Java/libraries/Firmata` (no Resources directory).*

#### 1.1.4.2 Windows:

1. Navigate to `c:/Program\ Files/arduino-1.x/libraries/` and replace the existing `Firmata` folder with the latest `Firmata release` (note there is a different download for Arduino 1.0.x vs 1.6.x).
2. Restart the Arduino application and the latest version of Firmata will be available.

*Update the path and Arduino version as necessary*

#### 1.1.4.3 Linux:

1. Navigate to `~/arduino-1.x/libraries/` and replace the existing `Firmata` folder with the latest `Firmata release` (note there is a different download for Arduino 1.0.x vs 1.6.x).
2. Restart the Arduino application and the latest version of Firmata will be available.

*Update the path and Arduino version as necessary*

#### 1.1.4.4 Using the Source code rather than release archive (only for versions older than Arduino 1.6.3)

*It is recommended you update to Arduino 1.6.4 or higher if possible, that way you can clone directly into the external Arduino/libraries/ directory which persists between Arduino application updates. Otherwise you will need to move your clone each time you update to a newer version of the Arduino IDE.*

If you're stuck with an older version of the IDE, then follow these keep reading otherwise jump up to the "Cloning Firmata section above".

Clone this repo directly into the core Arduino application libraries directory. If you are using Arduino 1.5.x or <= 1.6.3, the repo directory structure will not match the Arduino library format, however it should still compile as long as you are using Arduino 1.5.7 or higher.

You will first need to remove the existing Firmata library, then clone firmata/arduino into an empty Firmata directory:
```
$ rm -r /Applications/Arduino.app/Contents/Resources/Java/libraries/Firmata
$ git clone git@github.com:firmata/arduino.git
       /Applications/Arduino.app/Contents/Resources/Java/libraries/Firmata
```

*Update paths if you're using Windows or Linux*

To generate properly formatted versions of Firmata (for Arduino 1.0.x and Arduino 1.6.x), run the `release.sh` script.

### 1.1.5 Contributing

If you discover a bug or would like to propose a new feature, please open a new issue. Due to the limited memory of standard Arduino boards we cannot add every requested feature to StandardFirmata. Requests to add new features to StandardFirmata will be evaluated by the Firmata developers. However it is still possible to add new features to other Firmata implementations (Firmata is a protocol whereas StandardFirmata is just one of many possible implementations).

To contribute, fork this repository and create a new topic branch for the bug, feature or other existing issue you are addressing. Submit the pull request against the *master* branch.

If you would like to contribute but don't have a specific bugfix or new feature to contribute, you can take on an existing issue, see issues labeled "pull-request-encouraged". Add a comment to the issue to express your intent to begin work and/or to get any additional information about the issue.

You must thoroughly test your contributed code. In your pull request, describe tests performed to ensure that no existing code is broken and that any changes maintain backwards compatibility with the existing api. Test on multiple Arduino board variants if possible. We hope to enable some form of automated (or at least semi-automated) testing in the future, but for now any tests will need to be executed manually by the contributor and reviewers.

Use Artistic Style (astyle) to format your code. Set the following rules for the astyle formatter:
```
style = ""
indent-spaces = 2
indent-classes = true
indent-switches = true
indent-cases = true
indent-col1-comments = true
pad-oper = true
pad-header = true
keep-one-line-statements = true
```

If you happen to use Sublime Text, this astyle plugin is helpful. Set the above rules in the user settings file.

Chapter 2

# AccelStepperFirmata (Stepper 2.0)

Provides support for full 2 wire, full 3 wire, full 4 wire, half 3 wire, and half 4 wire stepper motor drivers (H-bridge, darlington array, etc) as well as step + direction drivers such as the `EasyDriver`. Current implementation supports 10 stepper motors at the same time (#[0-9]).

Includes optional support for acceleration and deceleration of the motor.

Also includes multiStepper support which allows groups of steppers to be simultaneously controlled. Up to five multiStepper groups can be created. The total number of steppers is still limited to 10.

AccelStepperFirmata sends and receives floats in a custom format described at the end of this document.

Example files:

- ConfigurableFirmata `AccelStepperFirmata.cpp`.

Added in Firmata protocol version 2.6.0.

## 2.1 Protocol

### Stepper configuration

This message is required and must be sent prior to any other message. The device number is arbitrary, but must be unique.

```
0  START_SYSEX                          (0xF0)
1  ACCELSTEPPER_DATA                    (0x62)
2  config command                       (0x00 = config)
3  device number                        (0-9) (Supports up to 10 motors)
4  interface                            (upper 3 bits = wire count:
                                          001XXXX = driver
                                          010XXXX = two wire
                                          011XXXX = three wire
                                          100XXXX = four wire)
                                        (4th - 6th bits = step type
                                          step size = 1/2^0bXXX
                                          Examples:
                                          XXX000X = whole step
                                          XXX001X = half step
                                          XXX010X = quarter step
                                          etc...)
                                        (lower 1 bit = has enable pin:
                                          XXXXXX0 = no enable pin
                                          XXXXXX1 = has enable pin)
5  motorPin1 or stepPin number          (0-127)
6  motorPin2 or directionPin number     (0-127)
```

```
7  [when interface >= 0x011] motorPin3        (0-127)
8  [when interface >= 0x100] motorPin4        (0-127)
9  [when interface && 0x0000001] enablePin    (0-127)
10 [optional] pins to invert                  (lower 5 bits = pins:
                                                XXXXXX1 = invert motorPin1
                                                XXXXX1X = invert motorPin2
                                                XXXX1XX = invert motorPin3
                                                XXX1XXX = invert motorPin4
                                                XX1XXXX = invert enablePin)
11 END_SYSEX                                  (0xF7)
```

### Stepper zero

accelStepper will store the current absolute position of the stepper motor (in steps). Sending the zero command will reset the position value to zero without moving the stepper.

```
0  START_SYSEX                             (0xF0)
1  ACCELSTEPPER_DATA                       (0x62)
2  zero command                            (0x01)
3  device number                           (0-9)
4  END_SYSEX                               (0xF7)
```

### Stepper step (relative move)

Steps to move is specified as a 32-bit signed long.

```
0  START_SYSEX                             (0xF0)
1  ACCELSTEPPER_DATA                       (0x62)
2  step command                            (0x02)
3  device number                           (0-9)
4  num steps, bits 0-6
5  num steps, bits 7-13
6  num steps, bits 14-20
7  num steps, bits 21-27
8  num steps, bits 28-32
9  END_SYSEX                               (0xF7)
```

### Stepper to (absolute move)

Moves a stepper to a desired position based on the number of steps from the zero position. Position is specified as a 32-bit signed long.

```
0  START_SYSEX                             (0xF0)
1  ACCELSTEPPER_DATA                       (0x62)
2  to command                              (0x03)
3  device number                           (0-9)
4  position, bits 0-6
5  position, bits 7-13
6  position, bits 14-20
7  position, bits 21-27
8  position, bits 28-32
9 END_SYSEX                               (0xF7)
```

### Stepper enable

For stepper motor controllers that are configured with an enable pin, the enable command manages whether the controller passes voltage through to the motor. When a stepper motor is idle, voltage is still being consumed so if the stepper motor does not need to hold its position use enable to save power.

```
0  START_SYSEX                             (0xF0)
1  ACCELSTEPPER_DATA                       (0x62)
2  enable command                          (0x04)
3  device number                           (0-9)
4  device state                            (HIGH : enabled | LOW : disabled)
5  END_SYSEX                               (0xF7)
```

### Stepper stop

Stops a stepper motor. Results in `STEPPER_MOVE_COMPLETE` being sent to the client with the position of the motor when stop is completed note: If an acceleration is set, stop will not be immediate.

```
0  START_SYSEX                             (0xF0)
1  ACCELSTEPPER_DATA                       (0x62)
2  stop command                            (0x05)
3  device number                           (0-9)
4  END_SYSEX                               (0xF7)
```

### Stepper report position (request)

Request a position report.
```
0  START_SYSEX                               (0xF0)
1  ACCELSTEPPER_DATA                         (0x62)
2  report position command                   (0x06)
3  device number                             (0-9)
4  END_SYSEX                                 (0xF7)
```

#### Stepper report position (reply)

Sent when a report position is requested. Position is reported as a 32-bit signed long.
```
0  START_SYSEX                               (0xF0)
1  ACCELSTEPPER_DATA                         (0x62)
2  report position command                   (0x06)
3  device number                             (0-9)
4  position, bits 0-6
5  position, bits 7-13
6  position, bits 14-20
7  position, bits 21-27
8  position, bits 28-31
9  END_SYSEX                                 (0xF7)
```

#### Stepper move complete

Sent when a move completes. Position is reported as a 32-bit signed long.
```
0  START_SYSEX                               (0xF0)
1  ACCELSTEPPER_DATA                         (0x62)
2  move complete command                     (0x0A)
3  device number                             (0-9)
4  position, bits 0-6
5  position, bits 7-13
6  position, bits 14-20
7  position, bits 21-27
8  position, bits 28-31
9  END_SYSEX                                 (0xF7)
```

#### Stepper limit

*Not yet implemented*

When a limit pin (digital) is set to its limit state, movement in that direction is disabled.
```
0  START_SYSEX                               (0xF0)
1  ACCELSTEPPER_DATA                         (0x62)
2  stop limit command                        (0x07)
3  device number                             (0-9)
4  lower limit pin number                    (0-127)
5  lower limit state                         (0x00 | 0x01)
6  upper limit pin number                    (0-127)
7  upper limit state                         (0x00 | 0x01)
8  END_SYSEX                                 (0xF7)
```

#### Stepper set acceleration

Sets the acceleration/deceleration in steps/sec^2. The accel value is passed using accelStepperFirmata's custom float format described below.
```
0  START_SYSEX                               (0xF0)
1  ACCELSTEPPER_DATA                         (0x62)
2  set acceleration command                  (0x08)
3  device number                             (0-9) (Supports up to 10 motors)
4  accel, bits 0-6                           (acceleration in steps/sec^2)
5  accel, bits 7-13
6  accel, bits 14-20
7  accel, bits 21-27
8  END_SYSEX                                 (0xF7)
```

#### Stepper set speed

If acceleration is off (equal to zero) sets the speed in steps per second. If acceleration is on (non-zero) sets the maximum speed in steps per second. The speed value is passed using accelStepperFirmata's custom float format described below.
```
0  START_SYSEX                               (0xF0)
1  ACCELSTEPPER_DATA                         (0x62)
2  set speed command                         (0x09)
3  device number                             (0-9) (Supports up to 10 motors)
4  maxSpeed, bits 0-6                        (maxSpeed in steps per sec)
5  maxSpeed, bits 7-13
```

```
6  maxSpeed, bits 14-20
7  maxSpeed, bits 21-27
8  END_SYSEX                               (0xF7)
```

#### MultiStepper configuration

Stepper instances that have been created with the stepper configuration command above can be added to a multi↵ Stepper group. Groups can be sent a list of devices/positions in a single command and their movements will be coordinated to begin and end simultaneously. Note that multiStepper does not support acceleration or deceleration.

```
0  START_SYSEX                            (0xF0)
1  ACCELSTEPPER_DATA                      (0x62)
2  multiConfig command                    (0x20)
3  group number                           (0-4)
4  member 0x00 device number              (0-9)
5  member 0x01 device number              (0-9)
6  [optional] member 0x02 device number   (0-9)
7  [optional] member 0x03 device number   (0-9)
8  [optional] member 0x04 device number   (0-9)
9  [optional] member 0x05 device number   (0-9)
10 [optional] member 0x06 device number   (0-9)
11 [optional] member 0x07 device number   (0-9)
12 [optional] member 0x08 device number   (0-9)
13 [optional] member 0x09 device number   (0-9)
14 END_SYSEX                              (0xF7)
```

#### MultiStepper to

Sets each stepper in a group to a desired position based on the number of steps from its zero position. Positions are specified as a 32-bit signed long.

Stepper movements will be coordinated so that all arrive at their desired position simultaneously. The duration of this move is based on which stepper will take the longest given the change in position and the stepper's max speed.

```
0  START_SYSEX                            (0xF0)
1  ACCELSTEPPER_DATA                      (0x62)
2  multi to command                       (0x21)
3  group number                           (0-4)
4  position, bits 0-6
5  position, bits 7-13
6  position, bits 14-20
7  position, bits 21-27
8  position, bits 28-31
*Repeat 4 through 8 for each device in group*
53 END_SYSEX                              (0xF7)
```

#### MultiStepper stop

Immediately stops all steppers in the group.

```
0  START_SYSEX                           (0xF0)
1  ACCELSTEPPER_DATA                     (0x62)
2  multi stop command                    (0x23)
3  group number                          (0-4)
4  END_SYSEX                             (0xF7)
```

#### MultiStepper move compelte

Sent when a multiStepper move completes.

```
0  START_SYSEX                           (0xF0)
1  ACCELSTEPPER_DATA                     (0x62)
2  multi stepper move complete command   (0x24)
3  group  number                         (0-4)
4  END_SYSEX                             (0xF7)
```

## 2.2 AccelStepperFirmata's Custom Float Format

Floats sent and received by accelStepperFirmata are composed of a 23-bit significand (mantissa) and a 4-bit, base 10 exponent (biased -11 with an explicit 1's bit) and a sign bit.

| 0-20 | 21 | 22-25 | 26-27 |
|---|---|---|---|
| least significant bits | sign | exponent | most significant bits |
| 21 bits | 1 bit | 4 bits | 2 bits |

These values allow a range from 8.388608∗10^-11 to 83886.08. Small enough to represent one step per year and large enough to exceed our max achievable stepper speed.

**Example 1: 1 step per hour**

1 step per hour = 1 step / 60 minutes / 60 seconds = 0.000277... steps per second

The largest integer that can be represented in 23 bits is 8388608 so the significand will be limited to 6 or 7 digits. In this case 2777777 (note the value truncates).

The exponent is 4 bits which limits the range to 0-15, but we subtract 11 from that value on the receiving end to give us a range from -11 to 4. In this example we are passing 1 to give us a -10 value in the exponent.

| | Decimal | Binary |
|---|---|---|
| Significand | 2777777 | 01010100110001010110001 |
| Exponent | 1 | 0001 |
| Sign | 0 | 0 |

Values in firmata are passed in the 7 least significant bits of each message byte so we will be passing in 4 bytes in this order:

| | Binary | Hex |
|---|---|---|
| Least most significant bits | 0110001 | 0x31 |
| Next most significant bits | 1000101 | 0x45 |
| Next most significant bits | 0101001 | 0x29 |
| Sign, Exponent and 2 most significant bits | 0000101 | 0x05 |

**Example 2: 100 steps per second**

We have to pad our significand on the right with four zeros to get our full precision. That makes the significand 100000000 and our exponent value will be 2. Since the value we send will be biased -11 on the receiving end, we send 13 in the exponent.

| | Decimal | Binary |
|---|---|---|
| Significand | 1 | 00000000000000000000001 |
| Exponent | 13 | 1101 |
| Sign | 0 | 0 |

Values in firmata are passed in the 7 least significant bits of each message byte so we would be passing in 4 bytes in this order:

| | Binary | Hex |
|---|---|---|
| Least most significant bits | 0000001 | 0x01 |
| Next most significant bits | 0000000 | 0x00 |
| Next most significant bits | 0000000 | 0x00 |
| Sign, Exponent and 2 most significant bits | 0110100 | 0x34 |

# Chapter 3

# encoder

#Encoder Feature

Provide incremental encoders support, for both `linear` and `rotary` encoders.

This feature is based on based on `PJRC's implementation`. See `this article` for more informations about how it works (well explained!).

Current implementation supports 5 encoders at the same time (#[0-4]) and you can activate automatic position reports every (SAMPLING_INTERVAL)ms. Reports are disabled by default.

For best Performances, connect only interrupt pins.

Added in Firmata protocol version 2.4.0.

Example files :

- EncoderFeature is a contributed feature for `ConfigurableFirmata.ino`.
- A dedicated example is available. See `SimpleEncoderFirmata.ino`.

### 3.0.1 Compatible client librairies

TODO : Update this

### 3.0.2 Tested boards

This feature has been tested on :

- Arduino Uno
- Arduino Mega
- Arduino Leonardo
- Arduino Due

### 3.0.3 Protocol details

The protocol below use exclusively SysEx queries and SysEx responses.

#### 3.0.3.1 Attach encoder query

Query :
```
/* -----------------------------------------------------
* 0 START_SYSEX                 (0xF0)
* 1 ENCODER_DATA                (0x61)
* 2 ENCODER_ATTACH              (0x00)
* 3 encoder #                   ([0 - MAX_ENCODERS-1])
* 4 pin A #                     (first pin)
* 5 pin B #                     (second pin)
* 6 END_SYSEX                   (0xF7)
* -----------------------------------------------------
*/
```

No response.

#### 3.0.3.2 Report encoder's position

Query
```
/* -----------------------------------------------------
* 0 START_SYSEX                 (0xF0)
* 1 ENCODER_DATA                (0x61)
* 2 ENCODER_REPORT_POSITION     (0x01)
* 3 Encoder #                   ([0 - MAX_ENCODERS-1])
* 4 END_SYSEX                   (0xF7)
* -----------------------------------------------------
*/
```

Response
```
/* -----------------------------------------------------
* 0 START_SYSEX                 (0xF0)
* 1 ENCODER_DATA                (0x61)
* 2 Encoder #  &  DIRECTION     [= (direction « 6) | (#)]
* 3 current position, bits 0-6
* 4 current position, bits 7-13
* 5 current position, bits 14-20
* 6 current position, bits 21-27
* 7 END_SYSEX                   (0xF7)
* -----------------------------------------------------
*/
```

Note : Byte #2 contains both encoder's number (i.e. channel) and encoder's direction. Direction is stored on the seventh bit, 0 (LOW) for positive and 1 (HIGH) for negative.
```
directionMask = 0x40; // B01000000
channelMask   = 0x3F; // B00111111
//ex direction is negative and encoder is on index 2
direction = 1;
channel = 2;
bytes[2] =  (direction « 6) | (channel);
```

#### 3.0.3.3 Report all encoders positions

Query
```
/* -----------------------------------------------------
* 0 START_SYSEX                 (0xF0)
* 1 ENCODER_DATA                (0x61)
* 2 ENCODER_REPORT_POSITIONS    (0x02)
* 3 END_SYSEX                   (0xF7)
* -----------------------------------------------------
*/
```

Response
```
/* -----------------------------------------------------
```

```
* 0 START_SYSEX                (0xF0)
* 1 ENCODER_DATA               (0x61)
* 2 first enc. #  & first enc. dir.
* 4 first enc. position, bits 0-6
* 5 first enc. position, bits 7-13
* 6 first enc. position, bits 14-20
* 7 first enc. position, bits 21-27
* 8 second enc. #  & second enc. dir.
* ...
* N END_SYSEX                  (0xF7)
* -----------------------------------------------------
*/
```

Note : `Report encoder's position` response is a special case of `Report all encoders positions` response.

#### 3.0.3.4 Reset encoder position to zero

Query

```
/* -----------------------------------------------------
* 0 START_SYSEX                (0xF0)
* 1 ENCODER_DATA               (0x61)
* 2 ENCODER_RESET_POSITION     (0x03)
* 3 encoder #                  ([0 - MAX_ENCODERS-1])
* 4 END_SYSEX                  (0xF7)
* -----------------------------------------------------
*/
```

No response.

#### 3.0.3.5 Enable/disable reporting

Query

```
/* -----------------------------------------------------
* 0 START_SYSEX                (0xF0)
* 1 ENCODER_DATA               (0x61)
* 2 ENCODER_REPORT_AUTO        (0x04)
* 3 enable                     (0x00 => false, true otherwise)
* 4 END_SYSEX                  (0xF7)
* -----------------------------------------------------
*/
```

No response.

Note : when reports are enabled, EncoderFirmata feature send the message below at every SAMPLING interval (19ms by default) :

```
/* -----------------------------------------------------
* 0 START_SYSEX                (0xF0)
* 1 ENCODER_DATA               (0x61)
* 2 first enc. #  & first enc. dir.   [= (direction « 6) | (#)]
* 4 first enc. position, bits 0-6
* 5 first enc. position, bits 7-13
* 6 first enc. position, bits 14-20
* 7 first enc. position, bits 21-27
* 8 second enc. #  & second enc. dir. [= (direction « 6) | (#)]
* ...
* N END_SYSEX                  (0xF7)
* -----------------------------------------------------
*/
```

#### 3.0.3.6 Detach encoder

Query

```
/* -----------------------------------------------------
* 0 START_SYSEX                (0xF0)
* 1 ENCODER_DATA               (0x61)
* 2 ENCODER_DETACH             (0x05)
* 3 encoder #                  ([0 - MAX_ENCODERS-1])
* 4 END_SYSEX                  (0xF7)
* -----------------------------------------------------
*/
```

No response.

Chapter 4

# Firmata sysex feature registry

The feature registry defines allocated and proposed Firmata feature IDs. The feature ID is the 2nd byte in the sysex message. An extended set of IDs is also available by setting the initial ID byte to 00H and then following with a 2 byte ID. All bytes between START_SYSEX and END_SYSEX must have the most significant bit set to 0.

| byte 0 | byte 1 | bytes 2 - N-1 | byte N |
|---|---|---|---|
| START_SYSEX | ID (01H-7DH) | PAYLOAD | END_SYSEX |
| START_SYSEX | ID (00H) | EXTENDED_ID (00H 00H - 7FH 7FH) + PAYLOAD | END_SYSEX |

## 4.1 Proposing a new feature

There are two different feature sets: Core features and optional features. See the descriptions for each type of feature set below. To propose a new core feature, open an issue to start a discussion. To propose a new optional feature, open an issue/and or a pull request adding a markdown file for the proposed feature. Also edit the optional feature set table to reserve an ID for the proposed feature and enter the status as "proposed". If the proposed feature exposes a very specific device or device driver (a NeoPixel light strip for example), assign an ID in the extended ID set (00H nnH nnH).

## 4.2 Core feature set

Core features are related to functionality such as digital and analog I/O, querying information about the state and capabilities of the microcontroller board and the firmware running on the board. The core features are documented in the protocol.md file and the full set of core features is versioned together using semver notation. The current protocol version is 2.5.1.

Firmata firmware should report the current protocol version (using the protocol version command↩
: 0xF9) and implement the full set of current core features defined for that version (with the exception of very limited hardware which can implement a subset of the core feature set).

*The range 01H - 0FH is reserved for user-defined features that are not added to this registry.*

| Feature ID | Feature name / link to documentation | Status |
|---|---|---|
| 00H | EXTENDED_ID | proposed |
| 01H - 0FH | *Reserved for user features* | n/a |
| 63H | REPORT_DIGTIAL_PIN - proposal | proposed |

| Feature ID | Feature name / link to documentation | Status |
|---|---|---|
| 64H | EXTENDED_REPORT_ANALOG - proposal | proposed |
| 65H | REPORT_FEATURES - proposal | proposed |
| 69H | ANALOG_MAPPING_QUERY | current |
| 6AH | ANALOG_MAPPING_RESPONSE | current |
| 6BH | CAPABILITY_QUERY | current |
| 6CH | CAPABILITY_RESPONSE | current |
| 6DH | PIN_STATE_QUERY | current |
| 6EH | PIN_STATE_RESPONSE | current |
| 6FH | EXTENDED_ANALOG | current |
| 71H | STRING_DATA | current |
| 79H | REPORT_FIRMWARE | current |
| 7AH | SAMPLING_INTERVAL | current |
| 7CH | ANALOG_CONFIG - proposal | proposed |
| 7EH | SYSEX_NON_REALTIME∗ | n/a |
| 7FH | SYSEX_REALTIME∗ | n/a |

∗∗7EH and 7FH are reserved because they have a special meaning to midi parsers.∗

## 4.3 Optional feature set

Optional features extend the hardware capabilities beyond basic digital I/O and analog I/O (eg: I2C, Serial/U↩ART, etc). Optional features also provide APIs to interface with general components (eg: servo, stepper, rotary encoder, etc) as well as specific components (eg: DHT11, NeoPixel, etc). The optional feature set also encompass functionality such as a general purpose scheduler API and a standardized device interface API. General features should use the single byte feature ID (allocating new IDs in descending order). However, any feature that wraps a specific driver, specific sensor, one-off custom component, etc should use the extended feature ID (`00H nnH nnH`) or should use the `DEVICE_QUERY/RESPONSE` API.

Each feature should be documented in a markdown file and versioned independently using semver notation. In the case where a feature spans multiple IDs (I2C for example), that entire set is documented in a single file and versioned together.

| Feature ID | Feature name | Link to documentation | Status |
|---|---|---|---|
| 5CH | RCOUTPUT_DATA | rcswitch-proposal.md | proposed |
| 5DH | RCINPUT_DATA | rcswitch-proposal.md | proposed |
| 5EH | DEVICE_QUERY | proposal | proposed |
| 5FH | DEVICE_RESPONSE | proposal | proposed |
| 60H | SERIAL_DATA (1.0) | serial-1.0.md | current |
| 61H | ENCODER_DATA | encoder.md | current |
| 62H | ACCELSTEPPER_DATA | accelStepperFirmata.md | current |
| 67H | SERIAL_DATA (2.0) | proposal | proposed |
| 68H | SPI_DATA | proposal | proposed |
| 70H | SERVO_CONFIG | servos.md | current |
| 72H | STEPPER_DATA | stepper-legacy.md | deprecated |
| 73H | ONEWIRE_DATA | onewire.md | current |
| 75H | SHIFT_DATA | shift-proposal.md | proposed |
| 76H | I2C_REQUEST | i2c.md | current |
| 77H | I2C_REPLY | i2c.md | current |
| 78H | I2C_CONFIG | i2c.md | current |

| Feature ID | Feature name | Link to documentation | Status |
|---|---|---|---|
| 7BH | SCHEDULER_DATA | scheduler.md | current |
| | | | |
| 00H nnH nnH | (start of extended feature ID set) | | |

# Chapter 5

# I2C

Enables communication with I2C devices. Currently only supports one I2C port per board.

Added in Firmata protocol version 2.1.0.

### 5.0.1 I2C read/write request

Updated in Firmata 2.5.1 to enable setting auto-restart by setting bit 6 of byte 3.

```
0  START_SYSEX (0xF0)
1  I2C_REQUEST (0x76)
2  slave address (LSB)
3  slave address (MSB) + read/write and address mode bits
     {bit 7: always 0}
     {bit 6: auto restart transmission, 0 = stop (default), 1 = restart}
     {bit 5: address mode, 1 = 10-bit mode}
     {bits 4-3: read/write, 00 = write, 01 = read once, 10 = read continuously, 11 = stop reading}
     {bits 2-0: slave address MSB in 10-bit mode, not used in 7-bit mode}
4  data 0 (LSB)
5  data 0 (MSB)
6  data 1 (LSB)
7  data 1 (MSB)
...
n  END_SYSEX (0xF7)
```

A note about read/write modes (above). The `read continuously` mode indicates that the firmware should continuously read the device at the rate specified by the `sampling interval`. A firmware implementation should support read continuous mode for several I2C devices simultaneously. Sending the `stop reading` command will end read continuous mode for that particular device.

*auto-restart (byte 3, bit 6) is needed by some devices such as the MMA8452Q accelerometer and the MPL3115As altimeter*

### 5.0.2 I2C reply

```
0  START_SYSEX (0xF0)
1  I2C_REPLY (0x77)
2  slave address (LSB)
3  slave address (MSB)
4  register (LSB)
5  register (MSB)
6  data 0 (LSB)
7  data 0 (MSB)
...
n  END_SYSEX (0XF7)
```

### 5.0.3 I2C config

```
0  START_SYSEX (0xF0)
1  I2C_CONFIG (0x78)
2  Delay in microseconds (LSB) [optional]
3  Delay in microseconds (MSB) [optional]
... user defined for special cases, etc
n  END_SYSEX (0xF7)
```

The optional `Delay` is for I2C devices that require a delay between when the register is written to and the data in that register can be read.

# Chapter 6

# OneWire

The idea is to configure Arduino Pins as OneWire Busmaster. The may be more than one pin configured for One↩ Wire and there may be more than one device connected to such a pin.

Each one-wire-device has a unique identifier which is 8 bytes long and comes factory-programmed into the the device. To scan all devices connected to a pin configured for onewire a SEARCH-request message is sent. The response contains all addresses of devices found. Having the address of a device OneWire-command-messages may be sent to this device.

The actual commands executed on the OneWire-bus are 'reset', 'skip', 'select', 'read', 'delay' and 'write' All these commands may be executed with a single OneWire-command-message. The subcommand-byte contains these commands bit-encoded. The data required to execute each bus-command must only be included in the message when the corresponding bit is set.

The order of execution of bus commands is: 'reset'->'skip'->'select'->'write'->'read'->'delay' (remember: each of these steps is optional. Also some combinations don't make sense and in fact are mutual exclusive in terms of OneWire bus protocol, so you cannot run a 'skip' followed by a 'select') The delay is useful for OneWire-commands included into taskdata (see `Firmata-scheduler proposal`).

Some OneWire-devices require some time to carry out e.g. a a/d-conversion after receiving the appropriate command. Including a delay into a OneWire-message saves some bytes in the taskdata (in comparism to the inclusion of a 'delay_task' scheduler message). OneWire Read- and ReadReply messages are correlated using a correlationid (16bits). The reply contains the correlationid-value that was sent with the original request.

Added in Firmata protocol version 2.4.0.

### 6.0.1 Example files:

- OneWire is include by default in `ConfigurableFirmata.ino`.
- `Example implementation` as a configurable Firmata feature class.

### 6.0.2 Compatible host implementations

- `ConfigurableFirmata`

### 6.0.3 Compatible client librairies

- perl-firmata
- node-firmata

### 6.0.4 Protocol details

OneWire SEARCH request
```
0  START_SYSEX       (0xF0)
1  OneWire Command   (0x73)
2  search command    (0x40|0x44) 0x40 normal search for all devices on the bus
                                 0x44 SEARCH_ALARMS request to find only those
                                 devices that are in alarmed state.
3  pin               (0-127)
4  END_SYSEX         (0xF7)
```

OneWire SEARCH reply
```
0  START_SYSEX       (0xF0)
1  OneWire Command   (0x73)
2  search reply command (0x42|0x45) 0x42 normal search reply
                                    0x45 reply to a SEARCH_ALARMS request
3  pin               (0-127)
4  bit 0-6   [optional] address bytes encoded using 8 times 7 bit for 7 bytes of 8 bit
5  bit 7-13  [optional] 1.address[0] = byte[0]    + byte[1]«7 & 0x7F
6  bit 14-20 [optional] 1.address[1] = byte[1]»1 + byte[2]«6 & 0x7F
7  ....                 ...
11 bit 49-55            1.address[6] = byte[6]»6 + byte[7]«1 & 0x7F
12 bit 56-63            1.address[7] = byte[8]    + byte[9]«7 & 0x7F
13 bit 64-69            2.address[0] = byte[9]»1 + byte[10]«6 &0x7F
n  ... as many bytes as needed (don't exceed MAX_DATA_BYTES though)
n+1  END_SYSEX       (0xF7)
```

OneWire CONFIG request
```
0  START_SYSEX       (0xF0)
1  OneWire Command   (0x73)
2  config command    (0x41)
3  pin               (0-127)
4  power             (0x00|0x01) 0x00 = leave pin on state high after write to support
                                 parasitic power
                                 0x01 = don't leave pin on high after write
5  END_SYSEX (0xF7)
```

OneWire COMMAND request
```
0  START_SYSEX       (0xF0)
1  OneWire Command   (0x73)
2  command bits      (0x00-0x2F) bit 0 = reset, bit 1 = skip, bit 2 = select,
                                 bit 3 = read, bit 4 = delay, bit 5 = write
3  pin               (0-127)
4  bit 0-6   [optional] data bytes encoded using 8 times 7 bit for 7 bytes of 8 bit
5  bit 7-13  [optional] data[0] = byte[0]   + byte[1]«7 & 0x7F
6  bit 14-20 [optional] data[1] = byte[1]>1 + byte[2]«6 & 0x7F
7  ....                 data[2] = byte = byte[2]>2 + byte[3]«5 & 0x7F ...
n  ... as many bytes as needed (don't exceed MAX_DATA_BYTES though)
n+1  END_SYSEX       (0xF7)
// data bytes within OneWire Request Command message
0  address[0]                     [optional, if bit 2 set]
1  address[1]                               "
2  address[2]                               "
3  address[3]                               "
4  address[4]                               "
5  address[5]                               "
6  address[6]                               "
7  address[7]                               "
8  number of bytes to read (LSB) [optional, if bit 3 set]
9  number of bytes to read (MSB)            "
10 request correlationid byte 0             "
11 request correlationid byte 1             "
10 delay in ms       (bits 0-7)   [optional, if bit 4 set]
11 delay in ms       (bits 8-15)            "
12 delay in ms       (bits 16-23)           "
13 delay in ms       (bits 24-31)           "
14 data to write     (bits 0-7)   [optional, if bit 5 set]
15 data to write     (bits 8-15)            "
16 data to write     (bits 16-23)           "
n  ... as many bytes as needed (don't exceed MAX_DATA_BYTES though)
```

### OneWire READ reply

```
0  START_SYSEX          (0xF0)
1  OneWire Command      (0x73)
2  read reply command   (0x43)
3  pin                  (0-127)
4  bit 0-6   [optional] data bytes encoded using 8 times 7 bit for 7 bytes of 8 bit
5  bit 7-13  [optional] correlationid[0] = byte[0]   + byte[1]«7 & 0x7F
6  bit 14-20 [optional] correlationid[1] = byte[1]>1 + byte[2]«6 & 0x7F
7  bit 21-27 [optional] data[0] = byte[2]>2 + byte[3]«5 & 0x7F
8  ....                 data[1] = byte[3]>3 + byte[4]«4 & 0x7F
n  ... as many bytes as needed (don't exceed MAX_DATA_BYTES though)
n+1  END_SYSEX          (0xF7)
```

# Chapter 7

# Digital Pin Groups (Proposal)

Provides support for the situation where you want to set or get the values of an arbitrary set of digital IO pins that may not necessarily align to a port and do all of this in one operation.

Examples of this behaviour would include:

- Analog Multiplexer / Demultimplexer, where you need to set the bit value of three pins in order to determine which analog line is being used on the multiplexer.
- Keypads where the value of the key presses are expressed using a combination of states across a set of digital lines (eg: `https://www.sparkfun.com/products/8653` )

When you want to issue an equivalent of a digitalWrite to a group of pins, you'll then issue a sequence of 7-bit bytes that provides the states of the pins collectively. This will save several calls to digital write and allow them to be done in one group.

Reads will work the same way but return a byte with the states of all of the pins. This is particularly important in a scenario like a keypad where independent async reads can make it extremely challenging to get the state of the keypress properly.

## 7.1 Requirements

- Currently unimplemented (PoC to come shortly)
- An array of pin groups (suggest 8 groups so it can be identifed with 3 bits each with up to 14 pins defined in the group)
- Modifications to firmata to accept the new protocol.

## 7.2 Protocol

### 7.2.1 Digital Pin Group commands

In order to save space in the protocol, the Digital Pin Group command comprises both protocol commands as well as the id address space for the groups as below:

LSB 0 - 2: 3 bits to determine which Pin Group command is being issued 3 : Reserved for future use / address space increases 4 - 6: 3 bits for Pin Group ID address space - providing up to 8 distinct digital pin groups

Command values are provided below
```
CONFIG              (0x00)
CLEAR               (0x01)
PIN_STATE_SET       (0x02)
PIN_STATE_REQUEST   (0x03)
PIN_STATE_REPLY     (0x04)
future reserved     (0x05 - 0x07)
```

### 7.2.2 Configuration

Specifies which pins should be grouped together and in which order. A maximum of 14 pins can be grouped together in one pin group. When specified in the config message, the pins will be provided in little endian style so the first pin will then be configured to mapped to the Least Significant Bit in subsequent write and read messages.

```
0:  START_SYSEX         (0xF0)
1:  pin group command   (0x60)
2:  pin group id (0 - 7) « 4 | CONFIG
3:  lowest bit set for pinMode (0=READ, 1=WRITE) top 6 bits reserved
4:  first pin in pin group (0 - 127)
5:  second pin in pin group (0 - 127)
... up to maximum of 14
N:  END_SYSEX           (0xF7)
```

### 7.2.3 Clear

Removes any pin entries associated to a pin group. This should free up any memory that has been allocted and remove any references to the pins that were configured. This is to ensure no side effects occur if a pin group is recycled.

```
0:  START_SYSEX         (0xF0)
1:  pin group command   (0x60)
2:  pin group id (0 -7) « 4 | CLEAR
3:  END_SYSEX           (0xF7)
```

### 7.2.4 State set

Sets the states of the pins in the group. As noted above, the first pin that is supplied in the config will be considered the least significant bit in this message. Any values provided that don't match the config definition should simply be ignored (ie a value comes through for the 5th pin in the group but none was defined so it should be ignored).

```
0:  START_SYSEX         (0xF0)
1:  pin group command   (0x60)
2:  pin group id (0 - 7) « 4 | PIN_STATE_SET
3:  packed 7 bit array as single byte providing values for the pin group
... optional second packed 7 bit array providing values for the pin group
N:  END_SYSEX           (0xF7)
```

### 7.2.5 State request and reply

Getting the states of the group of pins (essentially a group digital read) comprises a request to the board and a reply back.

Make a request for getting the states of the pin group.

```
0:  START_SYSEX         (0xF0)
1:  pin group command   (0x60)
2:  pin group id (0 - 7) « 4 | PIN_STATE_REQUEST
3:  END_SYSEX
```

Reply with the pin states.

```
0:  START_SYSEX         (0xF0)
1:  pin group command   (0x60)
2:  pin group id (0 - 7) « 4 | PIN_STATE_REPLY
3:  packed 7 bit array representing pin states, LSB is first pin defined in config
... optional second 7 bit array representing pin states for additional pins in group
N:  END_SYSEX
```

# Chapter 8

# RCSwitchFirmata Feature

`RCSwitchFirmata` is an adapter between `ConfigurableFirmata` and the `RCSwitch` library.

RCSwitch is a library to send and receive messages to/from radio controlled devices. Sender and receiver are referred to as *devices* within the context of this document. Multiple devices may be used at the same time; the only requirement is a pin per device. All devices may be used and configured independently. Thus, this document separates the main functions *send* and *receive*. RCSwitchFirmata is subdivided into *RCOutputFirmata* implementing the send function and *RCInputFirmata* implementing the receive function.

## 8.1 Protocol details

A common pattern of all queries is that they echo the query message as response. This pattern allows for detection of unsupported or wrong messages.

### 8.1.1 Tristate bits

RCSwitch supports - besides the types `long` and `char[]` - so-called *tristate* bits. A tristate bit has one of the values 0, 1, or F. Each tristate bit is coded as 2 bits as follows:

```
TRISTATE_0              0x00
TRISTATE_F              0x01
TRISTATE_RESERVED       0x02
TRISTATE_1              0x03
```

Thus, 1 byte consisting of 8 bits ABCDEFGH may hold up to 4 tristate bits AB, CD, EF and GH. The leftmost 2 bits represent the first tristate bit, the rightmost 2 bits represent the fourth tristate bit. If less than 4 tristate bits are used, the byte is filled with the reserved value `0x02`.

### 8.1.2 Send

#### 8.1.2.1 Attach sender

Query:

```
/*
 * -----------------------------------------------------
 * 0 START_SYSEX                (0xF0)
 * 1 RCOUTPUT_DATA              (0x5C)
 * 2 RCOUTPUT_ATTACH            (0x01)
 * 3 pin                        (pin number)
 * 4 END_SYSEX                  (0xF7)
 * -----------------------------------------------------
 */
```

Response: the query message

#### 8.1.2.2 Detach sender

Query:
```
/*
 * -----------------------------------------------------
 * 0 START_SYSEX                (0xF0)
 * 1 RCOUTPUT_DATA              (0x5C)
 * 2 RCOUTPUT_DETACH            (0x02)
 * 3 pin                        (pin number)
 * 4 END_SYSEX                  (0xF7)
 * -----------------------------------------------------
 */
```

Response: the query message

#### 8.1.2.3 Configuration of rcswitch parameter <tt>protocol</tt>

Query:
```
/*
 * -----------------------------------------------------
 * 0 START_SYSEX                (0xF0)
 * 1 RCOUTPUT_DATA              (0x5C)
 * 2 RCOUTPUT_PROTOCOL          (0x11)
 * 3 pin                        (pin number)
 * 4 protocol (int), bits 0-6
 * 5 protocol (int), bits 7-13
 * 6 protocol (int), bits 14-15
 * 7 END_SYSEX                  (0xF7)
 * -----------------------------------------------------
 */
```

Response: the query message

#### 8.1.2.4 Configuration of rcswitch parameter <tt>pulse length</tt>

Query:
```
/*
 * -----------------------------------------------------
 * 0 START_SYSEX                (0xF0)
 * 1 RCOUTPUT_DATA              (0x5C)
 * 2 RCOUTPUT_PULSE_LENGTH      (0x12)
 * 3 pin                        (pin number)
 * 4 pulse length (int), bits 0-6
 * 5 pulse length (int), bits 7-13
 * 6 pulse length (int), bits 14-15
 * 7 END_SYSEX                  (0xF7)
 * -----------------------------------------------------
 */
```

Response: the query message

#### 8.1.2.5 Configuration of rcswitch parameter <tt>repeat transmit</tt>

Query:
```
/*
 * -----------------------------------------------------
 * 0 START_SYSEX                (0xF0)
 * 1 RCOUTPUT_DATA              (0x5C)
 * 2 RCOUTPUT_PULSE_LENGTH      (0x14)
 * 3 pin                        (pin number)
 * 4 repeat transmit (int), bits 0-6
 * 5 repeat transmit (int), bits 7-13
 * 6 repeat transmit (int), bits 14-15
 * 7 END_SYSEX                  (0xF7)
 * -----------------------------------------------------
 */
```

Response: the query message

#### 8.1.2.6 Send tristate code as char array

Query:

```
/*
 * ------------------------------------------------------
 * 0 START_SYSEX                  (0xF0)
 * 1 RCOUTPUT_DATA                (0x5C)
 * 2 RCOUTPUT_TRISTATE            (0x21)
 * 3 pin                          (pin number)
 * 4..n-1 RC data (packed as 7-bit): char array with tristate bits ('0', '1', 'F')
 * n END_SYSEX                    (0xF7)
 * ------------------------------------------------------
 */
```

Response: the query message

#### 8.1.2.7 Send long code

Query:

```
/*
 * ------------------------------------------------------
 * 0 START_SYSEX                  (0xF0)
 * 1 RCOUTPUT_DATA                (0x5C)
 * 2 RCOUTPUT_CODE_LONG           (0x22)
 * 3 pin                          (pin number)
 * 4..n-1 RC data (packed as 7-bit): 2 bytes (int) with the number of bits to send, 4 bytes (long) data bits
 * n END_SYSEX                    (0xF7)
 * ------------------------------------------------------
 */
```

Response: the query message

#### 8.1.2.8 Send char array

Query:

```
/*
 * ------------------------------------------------------
 * 0 START_SYSEX                  (0xF0)
 * 1 RCOUTPUT_DATA                (0x5C)
 * 2 RCOUTPUT_CODE_CHAR           (0x24)
 * 3 pin                          (pin number)
 * 4..n-1 RC data (packed as 7-bit): char array with characters to send
 * n END_SYSEX                    (0xF7)
 * ------------------------------------------------------
 */
```

Response: the query message

#### 8.1.2.9 Send tristate code as packed tristate bits

Query:

```
/*
 * ------------------------------------------------------
 * 0 START_SYSEX                     (0xF0)
 * 1 RCOUTPUT_DATA                   (0x5C)
 * 2 RCOUTPUT_CODE_TRISTATE_PACKED   (0x28)
 * 3 pin                             (pin number)
 * 4..n-1 RC data (packed as 7-bit): byte array with 4 tristate bits per byte
 * n END_SYSEX                       (0xF7)
 * ------------------------------------------------------
 */
```

Response: the query message

### 8.1.3 Receive

#### 8.1.3.1 Attach receiver

Query:

```
/*
 * ------------------------------------------------------
 * 0 START_SYSEX                  (0xF0)
 * 1 RCINPUT_DATA                 (0x5D)
 * 2 RCINPUT_ATTACH               (0x01)
 * 3 pin                          (pin number)
 * 4 END_SYSEX                    (0xF7)
 * ------------------------------------------------------
 */
```

Response: the query message

#### 8.1.3.2 Detach receiver

Query:

```
/*
 * ------------------------------------------------------
 * 0 START_SYSEX                  (0xF0)
 * 1 RCINPUT_DATA                 (0x5D)
 * 2 RCINPUT_DETACH               (0x02)
 * 3 pin                          (pin number)
 * 4 END_SYSEX                    (0xF7)
 * ------------------------------------------------------
 */
```

Response: the query message

#### 8.1.3.3 Configuration of rcswitch parameter <tt>receive tolerance</tt> (in percent)

Query:

```
/*
 * ------------------------------------------------------
 * 0 START_SYSEX                  (0xF0)
 * 1 RCINPUT_DATA                 (0x5D)
 * 2 RCINPUT_TOLERANCE            (0x31)
 * 3 pin                          (pin number)
 * 4 tolerance                    (percent)
 * 5 END_SYSEX                    (0xF7)
 * ------------------------------------------------------
 */
```

Response: the query message

#### 8.1.3.4 Configuration parameter <tt>enable raw data</tt>

Query:

```
/*
 * ------------------------------------------------------
 * 0 START_SYSEX                  (0xF0)
 * 1 RCINPUT_DATA                 (0x5D)
 * 2 RCINPUT_ENABLE_RAW_DATA      (0x32)
 * 3 pin                          (pin number)
 * 4 rawdataEnabled               (0 for disabled, 1 for enabled)
 * 5 END_SYSEX                    (0xF7)
 * ------------------------------------------------------
 */
```

### 8.1.3.5 Receive a RF message

Query: none

Response:
```
/*
 * -----------------------------------------------------
 *  0 START_SYSEX                (0xF0)
 *  1 RCINPUT_DATA               (0x5D)
 *  2 RCINPUT_MESSAGE            (0x41)
 *  3 pin                        (pin number)
 *  4 message value (long), bits 0-6
 *  5 message value (long), bits 7-13
 *  6 message value (long), bits 14-20
 *  7 message value (long), bits 21-27
 *  8 message value (long), bits 28-32
 *  9 bitlength (int), bits 0-6
 * 10 bitlength (int), bits 7-13
 * 11 bitlength (int), bits 14-15
 * 12 delay (int), bits 0-6
 * 13 delay (int), bits 7-13
 * 14 delay (int), bits 14-15
 * 15 protocol (int), bits 0-6
 * 16 protocol (int), bits 7-13
 * 17 protocol (int), bits 14-15
 * 18..n-1 raw data (int[]); optional: only if rawdata was enabled
 * n END_SYSEX                  (0xF7)
 * -----------------------------------------------------
 */
```

# Chapter 9

# serial-2

#Serial 2.0 (Proposal)

Current version: 2.0.0

Enables control of up to 4 software and 4 hardware (UART) serial ports. Multiple ports can be used simultaneously (depending on restrictions of a given microcontroller board's capabilities).

Example files:

- Version 2.0 of the Serial feature has not yet been implemented.

## 9.0.1 Constants

### 9.0.1.1 Port IDs

Use these constants to identify the hardware or software serial port to address for each command.

```
HW_SERIAL0 = 0x00 (for using Serial when another transport is used for the Firmata Stream)
HW_SERIAL1 = 0x01
HW_SERIAL2 = 0x02
HW_SERIAL3 = 0x03
SW_SERIAL0 = 0x08
SW_SERIAL1 = 0x09
SW_SERIAL2 = 0x0A
SW_SERIAL3 = 0x0B
```

### 9.0.1.2 Serial pin capability response

Use these constants to identify the pin type in a `capability query response`.

```
// Where the pin mode = "Serial" and the pin resolution = one of the following:
RES_RX0 = 0x00
RES_TX0 = 0x01
RES_RX1 = 0x02
RES_TX1 = 0x03
RES_RX2 = 0x04
RES_TX2 = 0x05
RES_RX3 = 0x06
RES_TX3 = 0x07
// extensible up to 8 HW ports
```

### 9.0.1.3 Serial pin mode

```
PIN_MODE_SERIAL = 0x0A
```

### 9.0.2 Commands

#### 9.0.2.1 Serial Config

Configures the specified hardware or software serial port. RX and TX pins are optional and should only be specified if the platform requires them to be set.

```
0  START_SYSEX      (0xF0)
1  SERIAL_DATA      (0x67)  // command byte
2  SERIAL_CONFIG    (0x00)
3  port             (HW_SERIALn OR SW_SERIALn)
4  baud             (bits 0 - 6)
5  baud             (bits 7 - 13)
6  baud             (bits 14 - 20) // need to send 3 bytes for baud even if value is < 14 bits
7  rxPin            (0-127) [optional] // only set if platform requires RX pin number
8  txPin            (0-127) [optional] // only set if platform requires TX pin number
7|9 END_SYSEX       (0xF7)
```

#### 9.0.2.2 Serial Write

Firmata client -> Board

Receive serial data from Firmata client, reassemble and write for each byte received.

```
0  START_SYSEX      (0xF0)
1  SERIAL_DATA      (0x67)
2  SERIAL_WRITE     (0x01)
3  port             (HW_SERIALn OR SW_SERIALn)
4  data 0           (LSB)
5  data 0           (MSB)
6  data 1           (LSB)
7  data 1           (MSB)
...                 // up to max buffer - 5
n  END_SYSEX        (0xF7)
```

#### 9.0.2.3 Serial Read

Board -> Firmata client

Read contents of serial buffer and send to Firmata client (send with `SERIAL_REPLY`).

`maxBytesToRead` optionally specifies how many bytes to read for each iteration. Set to 0 (or do not define) to read all available bytes. If there are less bytes in the buffer than the number of bytes specified by `maxBytesTo↩ Read` then the lesser number of bytes will be returned.

```
0  START_SYSEX        (0xF0)
1  SERIAL_DATA        (0x67)
2  SERIAL_READ        (0x02)
3  port               (HW_SERIALn OR SW_SERIALn)
4  SERIAL_READ_MODE   (0x00) // 0x00 => read continuously, 0x01 => stop reading
5  maxBytesToRead     (lsb) [optional]
6  maxBytesToRead     (msb) [optional]
5|7 END_SYSEX         (0xF7)
```

#### 9.0.2.4 Serial Reply

Board -> Firmata client

Sent in response to a SERIAL_READ event or on each iteration of the reporting loop if `SERIAL_READ_CONTI↩ NUOUSLY` is set.

```
0  START_SYSEX        (0xF0)
1  SERIAL_DATA        (0x67)
2  SERIAL_REPLY       (0x03)
3  port               (HW_SERIALn OR SW_SERIALn)
4  data 0             (LSB)
5  data 0             (MSB)
6  data 1             (LSB)
7  data 1             (MSB)
...                   // up to max buffer - 5
n  END_SYSEX          (0xF7)
```

#### 9.0.2.5 Serial Close

Close the serial port. If you close a port, you will need to send a `SERIAL_CONFIG` message to reopen it.

```
0  START_SYSEX         (0xF0)
1  SERIAL_DATA         (0x67)
2  SERIAL_CLOSE        (0x04)
3  port                (HW_SERIALn OR SW_SERIALn)
4  END_SYSEX           (0xF7)
```

#### 9.0.2.6 Serial Flush

Flush the serial port. The exact behavior of flush depends on the underlying platform. For example, with Arduino, calling `flush` on a HW serial port will drain the TX output buffer, calling `flush` on a SW serial port will reset the RX buffer to the beginning, abandoning any data in the buffer. Other platforms may define `flush` differently as well so see the documentation of flush for the platform you are working with to understand exactly how it functions.

Generally `flush` is rarely needed so this functionality is primarily provided for advanced use cases.

```
0  START_SYSEX         (0xF0)
1  SERIAL_DATA         (0x67)
2  SERIAL_FLUSH        (0x05)
3  port                (HW_SERIALn OR SW_SERIALn)
4  END_SYSEX           (0xF7)
```

#### 9.0.2.7 Serial Listen

Enable switching serial ports. Necessary for Arduino SoftwareSerial but may not be applicable to other platforms.

```
0  START_SYSEX         (0xF0)
1  SERIAL_DATA         (0x67)
2  SERIAL_LISTEN       (0x06)
3  port                (HW_SERIALn OR SW_SERIALn)
4  END_SYSEX           (0xF7)
```

#### 9.0.2.8 Serial Update Baud

Update the baud rate on a configured serial port.

```
0  START_SYSEX         (0xF0)
1  SERIAL_DATA         (0x67)
2  SERIAL_UPDATE_BAUD  (0x07)
3  port                (HW_SERIALn OR SW_SERIALn)
4  baud                (bits 0 - 6)
5  baud                (bits 7 - 13)
6  baud                (bits 14 - 20) // need to send 3 bytes for baud even if value is < 14 bits
7  END_SYSEX           (0xF7)
```

#### 9.0.2.9 Serial Max Char Delay

Set to collect bytes received by serial port until the receive buffer is filled or a data gap is detected to avoid forwarding single bytes at baud rates below 50000.

To set a delay value, specify the number of bits, where the delay is calculated by the following:

numBits $*$ 1000 / baudRate

For example, if the baud is 9600, and 50 bits is specified (5 chars since 8N1 = 10 bits/char), then 50 $*$ 1000 / 9600 = 5.2 which is a delay of 5 milliseconds (value is char or int). This means approximately 5 chars will be sent every 5 milliseconds if the baud is 9600.

Ensure that numBits $*$ 1000 / baud is $>=$ 1.0 or serial data will be sent on every iteration.

A value of 0 = no delay (default behavior), results in single byte transfers to the host with baud rates below approximately 56k (varies with CPU speed).

```
0  START_SYSEX            (0xF0)
1  SERIAL_DATA            (0x67)
2  SERIAL_MAX_CHAR_DELAY  (0x08)
3  port                   (HW_SERIALn OR SW_SERIALn)
4  numBits                (0 - 127) // 50 is a good value for baud rates < 56k
7  END_SYSEX              (0xF7)
```

Chapter 10

# shift in/out proposal

There are a few different ways to approach shift in/out support. It's complicated since different hardware handles shift in/out in different ways. For example, not all hardware requires a latch pin and those that use some sort of a latch don't always use it the same way.

There has also been some discussion around supporting fractional byte devices. The proposals below do not include such functionality. I'm not sure how popular such devices are. If someone has a proposal that includes support for shifting fractional bytes, please submit a pull request to add the proposal to this document.

## 10.1 Propoasl A: shift in/out with no config or latch support

In this version it the user must configure the pin (input / output) separately. If the hardware they are using requires a latch, the latch pin must be managed separately.

```
// shift out
0  START_SYSEX
1  SHIFT_DATA          (0x75)
2  SHIFT_OUT           (0x01)
3  dataPin
4  clockPin
5  bitOrder            (MSBFIRST or LSBFIRST)
n ...                  (shift out data)
n+1 END_SYSEX
// shift in (for client application to request shift-in data from microcontroller)
0  START_SYSEX
1  SHIFT_DATA          (0x75)
2  SHIFT_IN            (0x02)
3  dataPin
4  clockPin
5  bitOrder            (MSBFIRST or LSBFIRST)
6  numBytes            (number of bytes to shift in. Default to 1)
7 END_SYSEX
// shift in reply (for sending shift-in data to client application)
0  START_SYSEX
1  SHIFT_DATA          (0x75)
2  SHIFT_IN_REPLY      (0x03)
3  dataPin             (so you know which data pin the reply corresponds to)
n ...                  (shift in data)
n+1 END_SYSEX
```

## 10.2 Proposal B: shift in/out with config and latch support

The advantages with this version over the one above is that pin modes are handled by the implementation (in the other version you have to handle them manually). You also send fewer bytes when shifting out or in data (since only have to specify clock, bitOrder and optional latch pin once when sending the config). The disadvantage is that memory must be allocated to store pin information.

Another advantage with this version is that you can rely on the firmware to do more of the work. For example you can shift in multiple bytes at a time and send them to the client in a single packet rather than a single byte at at time (if your hardware requires a latch/load pin).

This version uses a SHIFT_CONFIG command to set the clock pin, bitOrder and optional latchPin (or load for some shift-in hardware). There are a few different shift types / latch configurations:

```
// shift types (specified in bits 3:5 of byte 2)
SHIFT_OUT                     // shift out with no latch
SHIFT_IN                      // shift out with no latch/load
LATCH_L_SHIFT_OUT             // set latch low then shift out
LATCH_L_SHIFT_OUT_LATCH_H     // set latch low, shift out, then set latch high
SHIFT_OUT_LATCH_H             // shift out then set latch high
TOGGLE_LOAD_SHIFT_IN          // toggle load pin low, then high and shift in
```

The protocol is as follows:

```
// shift config (send when instantiating a new shift-based hardware module)
0  START_SYSEX
1  SHIFT_DATA    (0x75)
2  SHIFT_CONFIG  (bits 0:2 shift command, bits 3:5 shift type, bit 6 unused)
3  dataPin
4  clockPin
5  bitOrder
6  latchPin      [optional]
7 END_SYSEX
// shift out
0  START_SYSEX
1  SHIFT_DATA    (0x75)
2  SHIFT_OUT     (bits 0:2 shift command, bits 3:5 shift type, bit 6 unused)
3  dataPin
n  ...           (shift out data)
n+1 END_SYSEX
// shift in
0  START_SYSEX
1  SHIFT_DATA    (0x75)
2  SHIFT_IN      (bits 0:2 shift command, bits 3:5 shift type, bit 6 unused)
3  dataPin
4  numBytes      (number of bytes to shift in. Default to 1)
5  END_SYSEX
```

# Chapter 11

# SPI (Proposal)

A proposal for a SPI protocol for Firmata.

SPI is tricky to add to Firmata in a generic way since it is a fairly loose standard. There are variations in number of bits written and read, how the CS pin is used (if at all), use of other pins, etc. This proposal attempts to accommodate uses cases beyond the common sequence of:

1. set cs pin LOW
2. write/read 1 or more words
3. set cs Pin HIGH
4. return data read

### 11.0.1 Overview

A `SPI_BEGIN` command is used to initialize the SPI bus. Up to 8 SPI ports are supported using the `channel` parameter.

The `SPI_DEVICE_CONFIG` command is used to configure each attached SPI device.

There are 3 ways to send and receive data from the SPI slave device:

1. `SPI_TRANSFER` For each word written a word is read simultaneously.
2. `SPI_WRITE` Only write data (ignore any data returned by the slave device).
3. `SPI_READ` Only read data, writing `0` for each word to be read.

A `SPI_REPLY` is used to send requested data back to the client application when either a `SPI_TRANSFER` mode or `SPI_READ` command is sent.

A `SPI_END` command disables the SPI bus for the specified channel.

### 11.0.2 SPI_BEGIN

Required for platforms that require SPI bus initialization, such as Arduino. Optional if initialization is automatic (some Linux-based platforms for example).

Use `SPI_BEGIN` to initialize the SPI bus. Up to 8 SPI ports are supported, where each port is identified by a `channel` number (0-7).

`SPI_BEGIN` must be called at least once before sending any of the other commands.

`channel` is used to identify which SPI bus is used in the case that a board has multiple ports (SPI, SPI1, SPI2, etc). Many boards only have one port so the `channel` value will most often be `0`.

```
0:  START_SYSEX
1:  SPI_DATA              (0x68)
2:  SPI_BEGIN             (0x00)
3:  channel               (HW supports multiple SPI ports. range = 0-7, default = 0)
4:  END_SYSEX
```

### 11.0.3 SPI_DEVICE_CONFIG

Send this command once for each attached SPI device to initialize it before use. See parameter descriptions below.

```
0:  START_SYSEX
1:  SPI_DATA              (0x68)
2:  SPI_DEVICE_CONFIG     (0x01)
3:  deviceId | channel    (bits 3-6: deviceId, bits 0-2: channel)
4:  dataMode | bitOrder   (bits 1-2: dataMode (0-3), bit 0: bitOrder)
5:  maxSpeed              (bits 0 - 6)
6:  maxSpeed              (bits 7 - 14)
7:  maxSpeed              (bits 15 - 21)
8:  maxSpeed              (bits 22 - 28)
9:  maxSpeed              (bits 29 - 32)
10: wordSize              (0 = DEFAULT = 8-bits, 1 = 1-bit, 2 = 2 bits, etc)
11: csPinOptions          bit 0: CS_PIN_CONTROL (0 = disable
                                                 1 = enable (default))
                          bit 1: CS_ACTIVE_STATE (0 = Active LOW (default)
                                                  1 = Active HIGH)
                          bits 2-6: reserved for future options
12: csPin                 (0-127) The chip select pin number (ignored if
                          CS_PIN_CONTROL set to 0)
13: END_SYSEX
```

#### 11.0.3.1 deviceId

The `deviceId` may either be used as a specific identifier (Linux) or as an arbitrary identifier (Arduino). The `deviceId` value range is from 0 to 15 and can be specified separately for each SPI channel. The `deviceId` will also be returned with the channel for each `SPI_REPLY` command so it is clear which device the data corresponds to.

#### 11.0.3.2 bitOrder

```
LSBFIRST = 0
MSBFIRST = 1 (default)
```

#### 11.0.3.3 dataMode

| mode | clock polarity (CPOL) | clock phase (CPHA) |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

#### 11.0.3.4 maxSpeed

The maximum speed of communication with the SPI device. For a SPI device rated up to 5 MHz, use 5000000.

*For platforms that use a clock divider instead of a speed, pass the clock divider value instead.*

#### 11.0.3.5 wordSize

The size of a `word` in bits. Typically 8-bits (default). 0 = DEFAULT = 8-bits, 1 = 1 bit, 2 = 2 bits, etc (limit is TBD).

#### 11.0.3.6 csPinOptions / csPin

Use `CS_ACTIVE_STATE` to set the active state (typically LOW) for the CS pin. If the platform's SPI implementation does not already automatically handle the CS pin (it's handled automatically on Raspberry Pi, but not Arduino boards for example), then set `CS_PIN_CONTROL` to `enable` and specify the `csPin` number in byte 12. If the platform already handles the csPin then set `CS_PIN_CONTROL` to `disable` and the `csPin` number will be ignored (set to zero). For non-Linux platforms such as Arduino, to enable manual control of the CS pin using `DIGITAL_MES↩ SAGE` commands, set `CS_PIN_CONTROL` to `disable`.

### 11.0.4 SPI_TRANSFER

Full-duplex write/read transfer. This is the normal SPI transfer mode, a word must be written for every word read. Reply is sent via `SPI_REPLY`.

Since transport (Serial, Ethernet, Wi-Fi, etc) buffers tend to be small on microcontroller platforms, it may be necessary to send several `SPI_TRANSFER` commmands to complete a single SPI transaction. Use the `deselectCs↩ Pin` parameter to ensure the CS pin is not deselected in between `SPI_TRANSFER` commands until the transaction is complete.

`requestId` is used in the request messages `SPI_TRANSFER`, `SPI_WRITE` and `SPI_READ` and in the reply message `SPI_REPLY`. Its purpose is to ensure that the SPI_REPLY message matches the request. For each request message, increment a single 7-bit requestId value, rolling it over to 0 when $>$ 127.

`deselectCsPin` is used to control the csPin at the end of the transfer. By default the csPin will be deselected at the end of every transfer. However, to prevent deselection to enable back-to-back transfers for example, set `deselectCsPin` to `0` and the pin state won't be affected at the end of the transfer.

If `CS_PIN_CONTROL` is enabled, then the CS pin active state will be set when the `SPI_TRANSFER` command is received. It will only be deselected at the end of the transfer if `deselectCsPin` is set to 1.

```
0:  START_SYSEX
1:  SPI_DATA              (0x68)
2:  SPI_TRANSFER          (0x02)
3:  deviceId | channel    (bits 3-6: deviceId, bits 0-2: channel)
4:  requestId             (0-127) // increment for each call
5:  deselectCsPin         (0 = don't deselect csPin
                          1 = deselect csPin (default))
6.  numWords              (0-127: number of words to transfer)
7:  data 0                (bits 0-6)
8:  data 0                (bits 7-14 if word size if word size > 7 && < 15)
9:  data 0                (if word size > 14)
...                       up to numWords * (wordSize / 7)
N:  END_SYSEX
```

### 11.0.5 SPI_WRITE

Only write data, ignoring any data returned by the slave device.

Provided as a convenience. The same can be accomplished using `SPI_TRANSFER` and ignoring the `SPI_REPLY` command.

If `CS_PIN_CONTROL` is enabled, then the CS pin active state will be set when the `SPI_WRITE` command is received. It will only be deselected at the end of the write if `deselectCsPin` is set to 1.

A `SPI_WRITE` command should return a `SPI_REPLY` with a value of `1` if the write was successful or a value of `0` if the write failed.

```
0:  START_SYSEX
1:  SPI_DATA              (0x68)
2:  SPI_WRITE             (0x03)
3:  deviceId | channel    (bits 3-6: deviceId, bits 0-2: channel)
4:  requestId             (0-127) // increment for each call
5:  deselectCsPin         (0 = don't deselect csPin
                          1 = deselect csPin (default))
6.  numWords              (0-127: number of words to write)
7:  data 0                (bits 0-6)
8:  data 0                (bits 7-14 if word size if word size > 7 && < 15)
9:  data 0                (if word size > 14)
...                       up to numWords * (wordSize / 7)
N:  END_SYSEX
```

### 11.0.6 SPI_READ

Only read data, writing `0` for each word to be read. Reply is sent via `SPI_REPLY`.

Provided as a convenience. The same can be accomplished using `SPI_TRANSFER` and sending a `0` for each byte to be read.

If `CS_PIN_CONTROL` is enabled, then the CS pin active state will be set when the `SPI_READ` command is received. It will only be deselected at the end of the read if `deselectCsPin` is set to 1.

```
0:  START_SYSEX
1:  SPI_DATA              (0x68)
2:  SPI_WRITE             (0x04)
3:  deviceId | channel    (bits 3-6: deviceId, bits 0-2: channel)
4:  requestId             (0-127)  // increment for each call
5:  deselectCsPin         (0 = don't deselect csPin
                          1 = deselect csPin (default))
6.  numWords              (0-127: number of words to read)
7:  END_SYSEX
```

### 11.0.7 SPI_REPLY

An array of data received from the SPI slave device in response to a `SPI_TRANSFER` or `SPI_READ` command. The `requestId` should match the ID from the transfer, read or write command.

```
0:  START_SYSEX
1:  SPI_DATA              (0x68)
2:  SPI_REPLY             (0x05)
3:  deviceId | channel    (bits 3-6: deviceId, bits 0-2: channel)
4:  requestId             (0-127) // must match the ID from the request
5:  numWords              (0-127: number of words in the reply)
6:  data 0                (bits 0-6)
7:  data 0                (bits 7-14 if word size if word size > 7 && < 15)
8:  data 0                (if word size > 14)
...                       up to numWords * (wordSize / 7)
N:  END_SYSEX
```

### 11.0.8 SPI_END

Call to release SPI hardware send before quitting a Firmata client application.

```
0:  START_SYSEX
1:  SPI_DATA              (0x68)
2:  SPI_END               (0x06)
3:  channel               (HW supports multiple SPI ports. range = 0-7, default = 0)
4:  END_SYSEX
```

# Chapter 12

# tone proposal

Add ability to call Arduino `tone` and `noTone` functions. For non-Arduino architectures, similar functions to `tone` and `noTone` would need to be implemented.

The duration could be extended if necessary. Duration could also be optional. If left out, the user would need to send the NO_TONE command to stop the tone.

An implementation of this proposal is currently available `here`.

```
// wrapper for tone function
0  START_SYSEX        (0xF0)
1  TONE_DATA          (0x5F)
2  TONE               (0x00)
3  pinNumber
4  frequency LSB      (in HZ)
5  frequency MSB      (in HZ) (audible range is 20 HZ to 15,000 HZ so 14 bits is sufficient)
6  duration bits 0-6  (in ms)
7  duration bits 7-14 (in ms) (max duration is 16,383 milliseconds)
11  END_SYSEX (0xF7)
// wrapper for noTone function
0  START_SYSEX        (0xF0)
1  TONE_DATA          (0x5F)
2  NO_TONE            (0x01)
3  pinNumber
n  END_SYSEX          (0xF7)
```

# Chapter 13

# protocol

Current version: 2.6.0

The intention of this protocol is to allow as much of the microcontroller to be controlled as possible from the host computer. This protocol was designed for the direct communication between a microcontroller and a software object on a host computer. The host software object should then provide an interface that makes sense in that environment.

The data communication format uses MIDI messages. It is not necessarily a MIDI device, first it uses a faster serial speed, and second, the messages don't always map the same.

## 13.1 Message Types

This protocol uses the `MIDI message format`, but does not use the whole protocol. Most of the command mappings here will not be directly usable in terms of MIDI controllers and synths. It should co-exist with MIDI without trouble and can be parsed by standard MIDI interpreters. Just some of the message data is used differently.

| type | command | MIDI channel | first byte | second byte |
|---|---|---|---|---|
| analog I/O message | 0xE0 | pin # | LSB(bits 0-6) | MSB(bits 7-13) |
| digital I/O message | 0x90 | port | LSB(bits 0-6) | MSB(bits 7-13) |
| report analog pin | 0xC0 | pin # | disable/enable(0/1) | - n/a - |
| report digital port | 0xD0 | port | disable/enable(0/1) | - n/a - |
| | | | | |
| start sysex | 0xF0 | | | |
| set pin mode(I/O) | 0xF4 | | pin # (0-127) | pin mode |
| set digital pin value | 0xF5 | | pin # (0-127) | pin value(0/1) |
| sysex end | 0xF7 | | | |
| protocol version | 0xF9 | | major version | minor version |
| system reset | 0xFF | | | |

Sysex-based sub-commands (0x00 - 0x7F) are used for an extended command set.

| type | sub-command | first byte | second byte | ... |
|---|---|---|---|---|
| string | 0x71 | char $*$string ... | | |
| firmware name/version | 0x79 | major version | minor version | char $*$name ... |

## 13.2 Data Message Expansion

Two byte digital data format, second nibble of byte 0 gives the port number (eg 0x92 is the third port, port 2)

```
0  digital data, 0x90-0x9F, (MIDI NoteOn, bud different data format)
1  digital pins 0-6 bitmask
2  digital pin 7 bitmask
```

Analog 14-bit data format

```
0  analog pin, 0xE0-0xEF, (MIDI Pitch Wheel)
1  analog least significant 7 bits
2  analog most significant 7 bits
```

Version report format

```
0  version report header (0xF9) (MIDI Undefined)
1  major version (0-127)
2  minor version (0-127)
```

## 13.3 Control Messages Expansion

Set pin mode

```
0  set digital pin mode (0xF4) (MIDI Undefined)
1  set pin number (0-127)
2  mode (INPUT/OUTPUT/ANALOG/PWM/SERVO/I2C/ONEWIRE/STEPPER/ENCODER/SERIAL/PULLUP, 0/1/2/3/4/6/7/8/9/10/11)
```

Set digital pin value (added in v2.5)

```
0  set digital pin value (0xF5) (MIDI Undefined)
1  set pin number (0-127)
2  value (LOW/HIGH, 0/1)
```

Toggle analogIn reporting by pin

```
0  toggle analogIn reporting (0xC0-0xCF) (MIDI Program Change)
1  disable(0) / enable(non-zero)
```

*As of Firmata 2.4.0, upon enabling an analog pin, the pin value should be reported to the client application.*

Toggle digital port reporting by port (second nibble of byte 0), eg 0xD1 is port 1 is pins 8 to 15

```
0  toggle digital port reporting (0xD0-0xDF) (MIDI Aftertouch)
1  disable(0) / enable(non-zero)
```

*As of Firmata 2.4.0, upon enabling a digital port, the port value should be reported to the client application.*

Request version report

```
0  request version report (0xF9) (MIDI Undefined)
```

## 13.4 Sysex Message Format

System exclusive (sysex) messages are used to define sets of core and optional firmata features. Core features are related to functionality such as digital and analog I/O, querying information about the state and capabilities of the microcontroller board and the firmware running on the board. All core features are documented in this protocol.md file. Optional features extend the hardware capabilities beyond basic digital I/O and analog I/O and also provide APIs to interface with general and specific components and system services. Optional features are individually documented in separate markdown files.

Each firmata sysex message has a feature ID composed of either a single byte or an extended ID composed of 3 bytes where the first byte is always 0 to indicate it's an extended ID. The following table illustrates the structure. The most significant bit must be set to 0 in each byte between the `START_SYSEX` and `END_SYSEX` which frame the message.

| byte 0 | byte 1 | bytes 2 - N-1 | byte N |
|---|---|---|---|
| START_SYSEX | ID (01H-7DH) | PAYLOAD | END_SYSEX |
| START_SYSEX | ID (00H) | EXTENDED_ID (00H 00H - 7FH 7FH) + PAYLOAD | END_SYSEX |

Following are SysEx commands used for core features defined in this version of the protocol:

```
EXTENDED_ID                  0x00 // A value of 0x00 indicates the next 2 bytes define the extended ID
RESERVED                0x01-0x0F // IDs 0x01 - 0x0F are reserved for user defined commands
ANALOG_MAPPING_QUERY         0x69 // ask for mapping of analog to pin numbers
ANALOG_MAPPING_RESPONSE      0x6A // reply with mapping info
CAPABILITY_QUERY             0x6B // ask for supported modes and resolution of all pins
CAPABILITY_RESPONSE          0x6C // reply with supported modes and resolution
PIN_STATE_QUERY              0x6D // ask for a pin's current mode and state (different than value)
PIN_STATE_RESPONSE           0x6E // reply with a pin's current mode and state (different than value)
EXTENDED_ANALOG              0x6F // analog write (PWM, Servo, etc) to any pin
STRING_DATA                  0x71 // a string message with 14-bits per char
REPORT_FIRMWARE              0x79 // report name and version of the firmware
SAMPLING_INTERVAL            0x7A // the interval at which analog input is sampled (default = 19ms)
SYSEX_NON_REALTIME           0x7E // MIDI Reserved for non-realtime messages
SYSEX_REALTIME               0X7F // MIDI Reserved for realtime messages
```

The full set of core and optional Firmata feature IDs is defined in the `firmata-registry.md` file. See the registry for more info on proposing a new feature and obtaining an feature ID.

### 13.4.1 Query Firmware Name and Version

The firmware name to be reported should be exactly the same as the name of the Firmata client file, minus the file extension. So for StandardFirmata.ino, the firmware name is: StandardFirmata.

Query firmware Name and Version

```
0  START_SYSEX       (0xF0)
1  queryFirmware     (0x79)
2  END_SYSEX         (0xF7)
```

Receive Firmware Name and Version (after query)

```
0  START_SYSEX       (0xF0)
1  queryFirmware     (0x79)
2  major version     (0-127)
3  minor version     (0-127)
4  first char of firmware name (LSB)
5  first char of firmware name (MSB)
6  second char of firmware name (LSB)
7  second char of firmware name (MSB)
... for as many bytes as it needs
N  END_SYSEX         (0xF7)
```

### 13.4.2 Extended Analog

As an alternative to the normal analog message, this extended version allows addressing beyond pin 15 and supports sending analog values with any number of bits. The number of data bits is inferred by the length of the message.

```
0  START_SYSEX              (0xF0)
1  extended analog message  (0x6F)
2  pin                      (0-127)
3  bits 0-6                 (least significant byte)
4  bits 7-13                (most significant byte)
... additional bytes may be sent if more bits are needed
N  END_SYSEX                (0xF7)
```

### 13.4.3 Capability Query

The capability query provides a list of all modes supported by each pin. Each mode is described by 2 bytes where the first byte is the pin mode (such as digital input, digital output, PWM) and the second byte is the resolution (or sometimes the type of pin such as RX or TX for a UART pin). A value of `0x7F` is used as a separator to mark the end each pin's list of modes. The number of pins supported is inferred by the message length.

#### 13.4.3.1 Capabilities query

```
0  START_SYSEX              (0xF0)
1  CAPABILITY_QUERY         (0x6B)
2  END_SYSEX                (0xF7)
```

#### 13.4.3.2 Capabilities response

```
0  START_SYSEX              (0xF0)
1  CAPABILITY_RESPONSE      (0x6C)
2  1st supported mode of pin 0
3  1st mode's resolution of pin 0
4  2nd supported mode of pin 0
5  2nd mode's resolution of pin 0
... additional modes/resolutions, followed by `0x7F`,
    to mark the end of the pin's modes. Subsequently, each pin
    follows with its modes/resolutions and `0x7F`,
    until all pins are defined.
N  END_SYSEX                (0xF7)
```

**13.4.3.2.1 Supported Modes** The modes in the following list are the modes of operation that can be returned during the capability response:

```
DIGITAL_INPUT      (0x00)
DIGITAL_OUTPUT     (0x01)
ANALOG_INPUT       (0x02)
PWM                (0x03)
SERVO              (0x04)
SHIFT              (0x05)
I2C                (0x06)
ONEWIRE            (0x07)
STEPPER            (0x08)
ENCODER            (0x09)
SERIAL             (0x0A)
INPUT_PULLUP       (0x0B)
```

*If no modes are defined for a pin, no values are returned (other than the separator value `0x7F`) and it should be assumed that pin is unsupported by Firmata.*

**13.4.3.2.2 Mode Resolution** The resolution byte serves a couple of different purpose:

1. The original purpose was to define the resolution for analog input, pwm, servo and other modes that define a specific resolution (such as 10-bit for analog).
2. The resolution byte has been adapted for another purpose for Serial/UART pins, it defines if the pin is RX or TX and which UART it belongs to. `RX0` is the RX pin of UART0 (Serial on an Arduino for example), TX1 if the TX pin of UART1 (Serial1 on an Arduino).

Modes utilizing the resolution byte as resolution data:

```
DIGITAL_INPUT      (0x00) // resolution is 1 (binary)
DIGITAL_OUTPUT     (0x01) // resolution is 1 (binary)
ANALOG_INPUT       (0x02) // analog input resolution in number of bits
PWM                (0x03) // pwm resolution in number of bits
SERVO              (0x04) // servo resolution in number of bits
STEPPER            (0x08) // resolution is number number of bits in max number of steps
INPUT_PULLUP       (0x0B) // resolution is 1 (binary)
```

Modes utilizing the resolution byte to define type of pin:

```
SERIAL             (0x0A) // See description in
       [serial.md](https://github.com/firmata/protocol/blob/master/serial.md#serial-pin-capability-response)
// also to be added to I2C in the future to define SCL and SDA pins
```

*For other features (including I2C until updated) the resolution information is less important so a value of 1 is used.*

### 13.4.4 Analog Mapping Query

Analog messages are numbered 0 to 15, which traditionally refer to the Arduino pins labeled A0, A1, A2, etc. However, these pins are actually configured using "normal" pin numbers in the pin mode message, and when those pins are used for non-analog functions. The analog mapping query provides the information about which pins (as used with Firmata's pin mode message) correspond to the analog channels.

Analog mapping query
```
0  START_SYSEX              (0xF0)
1  analog mapping query     (0x69)
2  END_SYSEX                (0xF7)
```

Analog mapping response
```
0  START_SYSEX              (0xF0)
1  analog mapping response  (0x6A)
2  analog channel corresponding to pin 0, or 127 if pin 0 does not support analog
3  analog channel corresponding to pin 1, or 127 if pin 1 does not support analog
4  analog channel corresponding to pin 2, or 127 if pin 2 does not support analog
... etc, one byte for each pin
N  END_SYSEX                (0xF7)
```

*The above 2 queries provide static data (should never change for a particular board). Because this information is fixed and should only need to be read once, these messages are designed for a simple implementation in StandardFirmata, rather that bandwidth savings (eg, using packed bit fields).*

### 13.4.5 Pin State Query

The pin **state** is any data written to the pin (*it is important to note that pin state != pin value*). For output modes (digital output, PWM, and Servo), the state is any value that has been previously written to the pin. For input modes, typically the state is zero. However, for digital inputs, the state is the status of the pull-up resistor which is 1 if enabled, 0 if disabled.

The pin state query can also be used as a verification after sending pin modes or data messages.

Pin state query
```
0  START_SYSEX              (0xF0)
1  pin state query          (0x6D)
2  pin                      (0-127)
3  END_SYSEX                (0xF7)
```

Pin state response
```
0  START_SYSEX              (0xF0)
1  pin state response       (0x6E)
2  pin                      (0-127)
3  pin mode (the currently configured mode)
4  pin state, bits 0-6
5  (optional) pin state, bits 7-13
6  (optional) pin state, bits 14-20
... additional optional bytes, as many as needed
N  END_SYSEX                (0xF7)
```

### 13.4.6 String

Send short string messages between the board and the client application. String length is limited to half the buffer size - 3 (for Arduino this limits strings to 30 chars). Commonly used to report error messages to the client.
```
0  START_SYSEX        (0xF0)
1  STRING_DATA        (0x71)
2  first char LSB
3  first char MSB
4  second char LSB
5  second char MSB
... additional bytes up to half the buffer size - 3 (START_SYSEX, STRING_DATA, END_SYSEX)
N  END_SYSEX          (0xF7)
```

### 13.4.7 Sampling Interval

The sampling interval sets how often analog data and i2c data is reported to the client. The default for the arduino implementation is 19ms. This means that every 19ms analog data will be reported and any i2c devices with read continuous mode will be read.

```
0  START_SYSEX        (0xF0)
1  SAMPLING_INTERVAL  (0x7A)
2  sampling interval on the millisecond time scale (LSB)
3  sampling interval on the millisecond time scale (MSB)
4  END_SYSEX          (0xF7)
```

# Chapter 14

# Firmata Protocol Documentation

Firmata is a protocol for communicating with microcontrollers from software on a computer (or smartphone/tablet, etc). The protocol can be implemented in firmware on any microcontroller architecture as well as software on any computer software package (see list of client libraries below).

Firmata is based on the `midi message format` in that commands bytes are 8 bits and data bytes are 7 bits. For example the midi Channel Pressure (Command: 0xD0) message is 2 bytes long, in Firmata the Command 0xD0 is used to enable reporting for a digital port (collection of 8 pins). Both the midi and Firmata versions are 2 bytes long, but the meaning is obviously different. In Firmata, the number of bytes in a message must conform with the corresponding midi message. Midi `System Exclusive` (Sysex) messages however, can be any length and are therefore used most prominently throughout the Firmata protocol.

This repository contains documentation of the Firmata protocol. The core of the protocol is described in the protocol.md file file. Feature-specific documentation is described in individual markdown files (i2c.md, `accel↩ StepperFirmata.md`, servos.md, etc). Files added to the proposals directory are proposals for new features that have not yet been finalized. See `firmata-registry.md` for the full list of documented firmata features.

The Firmata protocol could theoretically be implemented for any microcontroller platform. Currently however, the most complete implementation is for `Arduino` (including Arduino-compatible microcontrollers). Here are the known Firmata microcontroller platform implementations:

- `Firmata for Arduino`
- `Firmata for Spark.io`

*Please note: I'm sure there are other implementations. If you know of others, please submit a pull request to update this readme file, or open an issue providing the link to be added to this document.*

## 14.1 Firmata client libraries

There are several client libraries. These are libraries that implement the Firmata protocol in order to communicate (from a computer, smartphone or tablet for example) with Firmata firmware running on a microcontroller platform. The following is a list of Firmata client library implementations:

- processing

  - [ https://github.com/firmata/processing]
  - [ http://funnel.cc]
- python
  - [ https://github.com/firmata/pyduino]
  - [ https://github.com/lupeke/python-firmata]
  - [ https://github.com/tino/pyFirmata]
  - [ https://github.com/MrYsLab/PyMata]
  - [ https://github.com/MrYsLab/pymata-aio]
- perl
  - [ https://github.com/ntruchsess/perl-firmata]
  - [ https://github.com/rcaputo/rx-firmata]
- ruby
  - [ https://github.com/hardbap/firmata]
  - [ https://github.com/PlasticLizard/rufinol]
  - [ http://funnel.cc]
- clojure
  - [ https://github.com/nakkaya/clodiuno]
  - [ https://github.com/peterschwarz/clj-firmata]
- javascript
  - [ https://github.com/jgautier/firmata]
  - [ http://breakoutjs.com]
  - [ https://github.com/rwldrn/johnny-five]
- java
  - [ https://github.com/4ntoine/Firmata]
  - [ https://github.com/kurbatov/firmata4j]
  - [ https://github.com/reapzor/FiloFirmata]
- .NET
  - [ https://github.com/SolidSoils/Arduino]
  - [ http://www.imagitronics.org/projects/firmatanet/]
  - [ https://github.com/wbadry/FirmataCSharpClientClass]
- Flash/AS3
  - [ http://funnel.cc]
  - [ http://code.google.com/p/as3glue/]
- PHP
  - [ https://bitbucket.org/ThomasWeinert/carica-firmata]
  - [ https://github.com/oasynnoum/phpmake_firmata]
- Haskell
  - [ http://hackage.haskell.org/package/hArduino]
- iOS

    - [ https://github.com/jacobrosenthal/iosfirmata]

- Dart
    - [ https://github.com/nfrancois/firmata]
- Max/MSP
    - [ http://www.maxuino.org/]
- Elixir
    - [ https://github.com/kfatehi/firmata]
- Modelica
    - [ https://www.wolfram.com/system-modeler/libraries/model-plug/]
- golang
    - [ https://github.com/kraman/go-firmata]
- Qt/QML
    - [ https://github.com/callaa/qfirmata]
- Android/Kotlin
    - [ https://github.com/xujiaao/android-firmata]
- Smalltalk
    - [ https://github.com/pharo-iot/Firmata]

*Each client library may not support the most recent version of the Firmata protocol and all features described in this reposity.*

## 14.2 Contributing

To submit a proposal for a new feature, create a `markdown` file for your proposal and append "-proposal" to the filename. Submit a pull request to add the proposal.

To make a change to an existing protocol, submit a pull request with your proposed changes. Be sure to provide any rationale in the pull request description.

Some hints for drafting a new proposal:

- See `feature-registry.md` for information on proposing a new feature and requesting a feature ID.
- Use sub-commands (3rd byte) as necessary if you have more than one message. See the `accelStepperFirmata.md` file for an example. Note the use of `0x62` for the feature ID and how each section has an enumerated set of subcommands (0x00 = config, 0x01 = zero, 02 = step, etc).
- It's okay to have optional values in a sysex message as long as those values are all at the end of the message. See the bytes 6 & 7 of the `SERIAL_CONFIG` message in `serial-1.0.md`
- Try to keep your sysex messages as short as possible.
- Pack bits if necessary. See the Response message for **Report encoder's position** in encoder.md for an example (also note how this was documented following the response message... please include similar documentation if you use bit packing in your proposal).
- If your proposal uses any of the available non-sysex midi messages, the number of bytes in the message must correspond to the number of bytes in the midi message. The meaning however does not need to be the same. However if the midi message uses channels (such as Note Off (0x80)) then the Firmata message must also use channels since a midi parser may expect this.

# Chapter 15

# Version 2.6.0 - September 16th, 2017

- Added AccelStepperFirmata (Stepper 2.0) for improved and more scalable stepper motor support.
- Deprecated the old Stepper protocol, now renamed to "stepper-legacy.md".

## 15.1 Version 2.5.1 - December 21st, 2015

- Enable I2C auto-restart by setting `bit 6 of byte 3` of the `I2C_REQUEST` message.

## 15.2 Version 2.5.0 - November 7th, 2015

- Added `Serial feature` for interfacing with serial devices via hardware or software serial.
- Added ability to set the value of a pin by sending a single pin value instead of a port value. See 'set digital pin value' in `protocol.md` for details.

## 15.3 Version 2.4.0 - December 2014

- Changed `report digital port` and `report analog pin` definition to return the port (digital) or pin (analog) value upon toggling to `enable`.
- Added `OneWire feature` to interface with 1-Wire devices.
- Added `Encoder feature` to interface with linear and rotary encoders.
- Added `Scheduler feature` to enable scheduling Firmata tasks. Useful when you need to send more data than the 64 byte serial buffer can handle.
- Added `Stepper feature` to enable interfacing with 2 wire and 4 wire stepper motor drivers and step + direction drivers.

*Note: The above 4 features were initially added for `ConfigurableFirmata` which had a different version number. They have been moved under the `v2.4.0` release here to get things back on track for the protocol version.*

## 15.4 Version 2.3.0 - February 2013

- Angle was removed from the `SERVO_CONFIG` message.

## 15.5 Version 2.2.0 - January 2011

- Added `Extended Analog` to allow addressing beyond pin 15 and support analog values with any number of bits.
- Added `Capability Query` to query the capabilities supported by each pin.
- Added `Analog Mapping Query` to map analog pin numbers to their digital pin number equivalent.
- Added `Pin State Query` to query the state of pin (output value or if input pullup enabled).

## 15.6 Version 2.1.0 - March 2010

- Added `I2C feature` to interface with I2C devices.
- Added `Servo feature` to interface with servo motors.
- Added ability to change the `sampling interval`.

## 15.7 Version 2.0.0 - September 2008

- Changed to 8-bit port-based digital messages (a collection of 8 pins) to mirror ports from previous 14-bit ports (a collection of 14 pins) modeled after the standard (ATmega8/168/328) Arduino boards.
- Added ability to `query firmware name and version`.

## 15.8 Version 1.0.0

- Switched to MIDI-compatible packet format (though the message interpretation differs).

# Chapter 16

# Scheduler

The idea is to store a stream of messages on a microcontroller which is replayed later (either once or repeated). A task is created by sending a create_task message. The time-to-run is initialized with 0 (which means the task is not yet ready to run). After filling up the taskdata with messages (using add_to_task command messages) a final schedule_task request is sent, that sets the time-to-run (in milliseconds after 'now'). If a task itself contains delay↩
_task or schedule_task-messages these cause the execution of the task to pause and resume after the amount of time given in such message has elapsed. If the last message in a task is a delay_task message the task is scheduled for reexecution after the amount of time specified. If there's no delay_task message at the end of the task (so the time-to-run is not updated during the run) the task gets deleted after execution.

Added in Firmata protocol version 2.4.0.

### 16.0.1 Example files:

- OneWire is include by default in `ConfigurableFirmata.ino`.
- `Example implementation` as a configurable Firmata feature class.

### 16.0.2 Compatible host implementations

- `ConfigurableFirmata`

### 16.0.3 Compatible client librairies

- `perl-firmata`

### 16.0.4 Protocol details

Scheduler CREATE_TASK request
```
0  START_SYSEX          (0xF0)
1  Scheduler Command    (0x7B)
2  create_task command  (0x00)
3  task id              (0-127)
4  length LSB           (bit 0-6)
5  length MSB           (bit 7-13)
6  END_SYSEX            (0xF7)
```

Scheduler DELETE_TASK request
```
0  START_SYSEX          (0xF0)
1  Scheduler Command    (0x7B)
2  delete_task command  (0x01)
3  task id              (0-127)
4  END_SYSEX            (0xF7)
```

Scheduler ADD_TO_TASK request
```
0  START_SYSEX          (0xF0)
1  Scheduler Command    (0x7B)
2  add_to_task command  (0x02)
3  task id              (0-127)
4  taskdata bit 0-6     [optional] task bytes encoded using 8 times 7 bit
                                   for 7 bytes of 8 bit
5  taskdata bit 7-13    [optional]
6  taskdata bit 14-20   [optional]
n  ... as many bytes as needed (don’t exceed MAX_DATA_BYTES though)
n+1  END_SYSEX          (0xF7)
```

Scheduler DELAY_TASK request
```
0  START_SYSEX          (0xF0)
1  Scheduler Command    (0x7B)
2  delay_task command   (0x03)
3  time_ms bit 0-6      time_ms is of type long, requires 32 bit.
4  time_ms bit 7-13
5  time_ms bit 14-20
6  time_ms bit 21-27
7  time_ms bit 28-31
8  END_SYSEX            (0xF7)
```

Scheduler SCHEDULE_TASK request
```
0  START_SYSEX              (0xF0)
1  Scheduler Command        (0x7B)
2  schedule_task command    (0x04)
3  task id                  (0-127)
4  time_ms bit 0-6          time_ms is of type long, requires 32 bit.
5  time_ms bit 7-13
6  time_ms bit 14-20
7  time_ms bit 21-27
8  time_ms bit 28-31
9  END_SYSEX                (0xF7)
```

Scheduler QUERY_ALL_TASKS request
```
0  START_SYSEX              (0xF0)
1  Scheduler Command        (0x7B)
2  query_all_tasks command  (0x05)
3  END_SYSEX                (0xF7)
```

Scheduler QUERY_ALL_TASKS reply
```
0  START_SYSEX          (0xF0)
1  Scheduler Command    (0x7B)
2  query_all_tasks Reply Command (0x09)
3  taskid_1             (0-127) [optional]
4  taskid_2             (0-127) [optional]
n  ... as many bytes as needed (don’t exceed MAX_DATA_BYTES though)
n+1  END_SYSEX (0xF7)
```

Scheduler QUERY_TASK request
```
0  START_SYSEX              (0xF0)
1  Scheduler Command        (0x7B)
2  query_task command       (0x06)
3  task id                  (0-127)
4  END_SYSEX                (0xF7)
```

Scheduler QUERY_TASK reply
```
0  START_SYSEX          (0xF0)
1  Scheduler Command    (0x7B)
```

```
2  query_task Reply Commandc (0x0A)
3  task id              (0-127)
4  time_ms bit 0-6
5  time_ms bit 7-13
6  time_ms bit 14-20
7  time_ms bit 21-27
8  time_ms bit 28-31 | (length bit 0-2) « 4
9  length bit 3-9
10 length bit 10-15 | (position bit 0) « 7
11 position bit 1-7
12 position bit 8-14
13 position bit 15 | taskdata bit 0-5 « 1 [taskdata is optional]
14 taskdata bit 6-12  [optional]
15 taskdata bit 13-19 [optional]
n  ... as many bytes as needed (don't exceed MAX_DATA_BYTES though)
n+1  END_SYSEX          (0xF7)
```

### Scheduler RESET request

```
0  START_SYSEX              (0xF0)
1  Scheduler Command        (0x7B)
2  scheduler reset command  (0x07)
3  END_SYSEX                (0xF7)
```

### Scheduler ERROR_FIRMATA_TASK reply

```
0  START_SYSEX              (0xF0)
1  Scheduler Command        (0x7B)
2  error_task Reply Command (0x08)
3  task id                  (0-127)
4  time_ms bit 0-6
5  time_ms bit 7-13
6  time_ms bit 14-20
7  time_ms bit 21-27
8  time_ms bit 28-31 | (length bit 0-2) « 4
9  length bit 3-9
10 length bit 10-15 | (position bit 0) « 7
11 position bit 1-7
12 position bit 8-14
13 position bit 15 | taskdata bit 0-5 « 1 [taskdata is optional]
14 taskdata bit 6-12  [optional]
15 taskdata bit 13-19 [optional]
n  ... as many bytes as needed (don't exceed MAX_DATA_BYTES though)
n+1  END_SYSEX              (0xF7)
```

# Chapter 17

# serial-1

#Serial 1.0

Enables control of up to 4 software and 4 hardware (UART) serial ports. Multiple ports can be used simultaneously (depending on restrictions of a given microcontroller board's capabilities).

Sample implementation code for Arduino is available `here`.

A client implementation can be found `here`.

Added in Firmata protocol version 2.5.0

## 17.0.1 Constants

### 17.0.1.1 Port IDs

Use these constants to identify the hardware or software serial port to address for each command.

```
HW_SERIAL0 = 0x00 (for using Serial when another transport is used for the Firmata Stream)
HW_SERIAL1 = 0x01
HW_SERIAL2 = 0x02
HW_SERIAL3 = 0x03
// extensible up to 8 HW serial ports
SW_SERIAL0 = 0x08
SW_SERIAL1 = 0x09
SW_SERIAL2 = 0x0A
SW_SERIAL3 = 0x0B
// extensible up to 8 SW serial ports
```

### 17.0.1.2 Serial pin capability response

Use these constants to identify the pin type in a `capability query response`.

```
// Where the pin mode = "Serial" and the pin resolution = one of the following:
RES_RX0 = 0x00
RES_TX0 = 0x01
RES_RX1 = 0x02
RES_TX1 = 0x03
RES_RX2 = 0x04
RES_TX2 = 0x05
RES_RX3 = 0x06
RES_TX3 = 0x07
// extensible up to 8 HW ports
```

### 17.0.1.3 Serial pin mode

```
PIN_MODE_SERIAL = 0x0A
```

### 17.0.2 Commands

#### 17.0.2.1 Serial Config

Configures the specified hardware or software serial port. RX and TX pins are optional and should only be specified if the platform requires them to be set.

```
0  START_SYSEX      (0xF0)
1  SERIAL_DATA      (0x60)  // command byte
2  SERIAL_CONFIG    (0x10)  // OR with port (0x11 = SERIAL_CONFIG | HW_SERIAL1)
3  baud             (bits 0 - 6)
4  baud             (bits 7 - 13)
5  baud             (bits 14 - 20) // need to send 3 bytes for baud even if value is < 14 bits
6  rxPin            (0-127) [optional] // only set if platform requires RX pin number
7  txPin            (0-127) [optional] // only set if platform requires TX pin number
6|8 END_SYSEX      (0xF7)
```

#### 17.0.2.2 Serial Write

Firmata client -> Board

Receive serial data from Firmata client, reassemble and write for each byte received.

```
0  START_SYSEX      (0xF0)
1  SERIAL_DATA      (0x60)
2  SERIAL_WRITE     (0x20) // OR with port (0x21 = SERIAL_WRITE | HW_SERIAL1)
3  data 0           (LSB)
4  data 0           (MSB)
5  data 1           (LSB)
6  data 1           (MSB)
...                 // up to max buffer - 5
n  END_SYSEX        (0xF7)
```

#### 17.0.2.3 Serial Read

Board -> Firmata client

Read contents of serial buffer and send to Firmata client (send with `SERIAL_REPLY`).

`maxBytesToRead` optionally specifies how many bytes to read for each iteration. Set to 0 (or do not define) to read all available bytes. If there are less bytes in the buffer than the number of bytes specified by `maxBytesTo↩ Read` then the lesser number of bytes will be returned.

```
0  START_SYSEX        (0xF0)
1  SERIAL_DATA        (0x60)
2  SERIAL_READ        (0x30) // OR with port (0x31 = SERIAL_READ | HW_SERIAL1)
3  SERIAL_READ_MODE   (0x00) // 0x00 => read continuously, 0x01 => stop reading
4  maxBytesToRead     (lsb) [optional]
5  maxBytesToRead     (msb) [optional]
4|6 END_SYSEX         (0xF7)
```

#### 17.0.2.4 Serial Reply

Board -> Firmata client

Sent in response to a SERIAL_READ event or on each iteration of the reporting loop if `SERIAL_READ_CONTI↩ NUOUSLY` is set.

```
0  START_SYSEX        (0xF0)
1  SERIAL_DATA        (0x60)
2  SERIAL_REPLY       (0x40) // OR with port (0x41 = SERIAL_REPLY | HW_SERIAL1)
3  data 0             (LSB)
4  data 0             (MSB)
3  data 1             (LSB)
4  data 1             (MSB)
...                   // up to max buffer - 5
n  END_SYSEX          (0xF7)
```

#### 17.0.2.5 Serial Close

Close the serial port. If you close a port, you will need to send a `SERIAL_CONFIG` message to reopen it.

```
0  START_SYSEX        (0xF0)
1  SERIAL_DATA        (0x60)
2  SERIAL_CLOSE       (0x50) // OR with port (0x51 = SERIAL_CLOSE | HW_SERIAL1)
3  END_SYSEX          (0xF7)
```

#### 17.0.2.6 Serial Flush

Flush the serial port. The exact behavior of flush depends on the underlying platform. For example, with Arduino, calling `flush` on a HW serial port will drain the TX output buffer, calling `flush` on a SW serial port will reset the RX buffer to the beginning, abandoning any data in the buffer. Other platforms may define `flush` differently as well so see the documentation of flush for the platform you are working with to understand exactly how it functions.

Generally `flush` is rarely needed so this functionality is primarily provided for advanced use cases.

```
0  START_SYSEX        (0xF0)
1  SERIAL_DATA        (0x60)
2  SERIAL_FLUSH       (0x60) // OR with port (0x61 = SERIAL_FLUSH | HW_SERIAL1)
3  END_SYSEX          (0xF7)
```

#### 17.0.2.7 Serial Listen

Enable switching serial ports. Necessary for Arduino SoftwareSerial but may not be applicable to other platforms.

```
0  START_SYSEX        (0xF0)
1  SERIAL_DATA        (0x60)
2  SERIAL_LISTEN      (0x70) // OR with port to switch to (0x79 = switch to SW_SERIAL1)
3  END_SYSEX          (0xF7)
```

# Chapter 18

# Servo

Send a Servo config message to configure a pin a servo. Then use the `SET_PIN_MODE` message to attach/detach Servo support to a pin. This saves space in the protocol by reusing the `SET_PIN_MODE` message, but the host software implementation could have a different interface, e.g. Arduino's `attach()` and `detach()`.

The `SERVO_CONFIG` message can be sent at any time to chang the settings.

Added in Firmata protocol version 2.1.0.

Servo config
```
// minPulse and maxPulse are 14-bit unsigned integers
0  START_SYSEX          (0xF0)
1  SERVO_CONFIG         (0x70)
2  pin number           (0-127)
3  minPulse LSB         (0-6)
4  minPulse MSB         (7-13)
5  maxPulse LSB         (0-6)
6  maxPulse MSB         (7-13)
7  END_SYSEX            (0xF7)
```

*This is just the standard* `SET_PIN_MODE` *message:* Set digital pin mode
```
0  set digital pin mode (0xF4) (MIDI Undefined)
1  pin number           (0-127)
2  state                (SERVO, 4)
```

Write to servo, servo write is performed if the pin mode is SERVO
```
0  ANALOG_MESSAGE       (0xE0-0xEF)
1  value LSB
2  value MSB
```

If the pin number is higher than 15, or if the value to write to the servo is greater than 14 bits, then the Extended Analog message can be used in place of the standard `ANALOG_MESSAGE`:
```
0  START_SYSEX              (0xF0)
1  extended analog message  (0x6F)
2  pin                      (0-127)
3  bits 0-6                 (least significant byte)
4  bits 7-13                (most significant byte)
... additionaly bytes may be sent if more bits are needed
N  END_SYSEX                (0xF7)
```

# Chapter 19

# Stepper Motor

*Note: This legacy version is deprecated as of Firmata protocol v2.6.0 and therefore is not recommended for new implementations. Please use the new, more full-featured* `AccelStepper version` *instead.*

Provides support for 4 wire and 2 wire stepper motor drivers (H-bridge, darlington array, etc) as well as step + direction drivers such as the `EasyDriver`. Current implementation supports 6 stepper motors at the same time (#[0-5]).

Also includes optional support for acceleration and deceleration of the motor.

Added in Firmata protocol version 2.4.0. Deprecated in Firmata protocol version 2.6.0.

Example files:

- The Stepper feature is include by default in `ConfigurableFirmata.ino`.
- `Example implementation` as a configurable Firmata feature class.
- `Example of Stepper implementation in StandardFirmata`. *Note the dependency on the FirmataStepper class.*

## 19.1 Protocol

Stepper configuration

*Note:* `stepDelay` *is the the number of microseconds between steps. The default value is 1us. You can change the delay to 2us (useful for high current stepper motor drivers). Additional delay values can be added in the future.*

```
0  START_SYSEX                      (0xF0)
1  Stepper Command                  (0x72)
2  config command                   (0x00 = config, 0x01 = step)
3  device number                    (0-5) (supports up to 6 motors)
4  stepDelay | interface            (upper 4 bits = step delay:
                                       0000XXX = default 1us delay [default]
                                       0001XXX = 2us delay
                                       additional bits not yet used)
                                    (lower 3 bits = interface:
                                       XXXX001 = step + direction driver
                                       XXXX010 = two wire
                                       XXXX100 = four wire)
5  steps-per-revolution LSB
6  steps-per-revolution MSB
7  motorPin1 or directionPin number  (0-127)
8  motorPin2 or stepPin number       (0-127)
9  [only when interface = 0x04] motorPin3 (0-127)
10 [only when interface = 0x04] motorPin4 (0-127)
11 END_SYSEX                        (0xF7)
```

### Stepper step

```
0  START_SYSEX          (0xF0)
1  Stepper Command      (0x72)
2  config command       (0x01)
3  device number        (0-5)
4  direction            (0-1) (0x00 = CW, 0x01 = CCW)
5  num steps byte1 LSB
6  num steps byte2
7  num steps byte3 MSB  (21 bits (2,097,151 steps max))
8  speed LSB            (steps in 0.01*rad/sec  (2050 = 20.50 rad/sec))
9  speed MSB
10 [optional] accel LSB (acceleration in 0.01*rad/sec^2 (1000 = 10.0 rad/sec^2))
11 [optional] accel MSB
12 [optional] decel LSB (deceleration in 0.01*rad/sec^2)
13 [optional] decel MSB
14 END_SYSEX            (0xF7)
```

# Chapter 20

# Firmata_Documentation

# Chapter 21

# Namespace Index

## 21.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 22

# Hierarchical Index

## 22.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 23

# Data Structure Index

## 23.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 24

# File Index

## 24.1 File List

Here is a list of all files with brief descriptions:

# Chapter 25

# Namespace Documentation

## 25.1 firmata Namespace Reference

### Data Structures

- class FirmataClass
- class FirmataMarshaller
- class FirmataParser

### Variables

- static const int FIRMWARE_MAJOR_VERSION = 2
- static const int FIRMWARE_MINOR_VERSION = 5
- static const int FIRMWARE_BUGFIX_VERSION = 7
- static const int PROTOCOL_MAJOR_VERSION = 2
- static const int PROTOCOL_MINOR_VERSION = 5
- static const int PROTOCOL_BUGFIX_VERSION = 1
- static const int MAX_DATA_BYTES = 64
- static const int DIGITAL_MESSAGE = 0x90
- static const int ANALOG_MESSAGE = 0xE0
- static const int REPORT_ANALOG = 0xC0
- static const int REPORT_DIGITAL = 0xD0
- static const int SET_PIN_MODE = 0xF4
- static const int SET_DIGITAL_PIN_VALUE = 0xF5
- static const int REPORT_VERSION = 0xF9
- static const int SYSTEM_RESET = 0xFF
- static const int START_SYSEX = 0xF0
- static const int END_SYSEX = 0xF7
- static const int SERIAL_DATA = 0x60
- static const int ENCODER_DATA = 0x61
- static const int SERVO_CONFIG = 0x70
- static const int STRING_DATA = 0x71
- static const int STEPPER_DATA = 0x72
- static const int ONEWIRE_DATA = 0x73
- static const int SHIFT_DATA = 0x75
- static const int I2C_REQUEST = 0x76
- static const int I2C_REPLY = 0x77

- static const int I2C_CONFIG = 0x78
- static const int REPORT_FIRMWARE = 0x79
- static const int EXTENDED_ANALOG = 0x6F
- static const int PIN_STATE_QUERY = 0x6D
- static const int PIN_STATE_RESPONSE = 0x6E
- static const int CAPABILITY_QUERY = 0x6B
- static const int CAPABILITY_RESPONSE = 0x6C
- static const int ANALOG_MAPPING_QUERY = 0x69
- static const int ANALOG_MAPPING_RESPONSE = 0x6A
- static const int SAMPLING_INTERVAL = 0x7A
- static const int SCHEDULER_DATA = 0x7B
- static const int SYSEX_NON_REALTIME = 0x7E
- static const int SYSEX_REALTIME = 0x7F
- static const int PIN_MODE_INPUT = 0x00
- static const int PIN_MODE_OUTPUT = 0x01
- static const int PIN_MODE_ANALOG = 0x02
- static const int PIN_MODE_PWM = 0x03
- static const int PIN_MODE_SERVO = 0x04
- static const int PIN_MODE_SHIFT = 0x05
- static const int PIN_MODE_I2C = 0x06
- static const int PIN_MODE_ONEWIRE = 0x07
- static const int PIN_MODE_STEPPER = 0x08
- static const int PIN_MODE_ENCODER = 0x09
- static const int PIN_MODE_SERIAL = 0x0A
- static const int PIN_MODE_PULLUP = 0x0B
- static const int PIN_MODE_IGNORE = 0x7F
- static const int TOTAL_PIN_MODES = 13

## 25.1.1 Variable Documentation

#### 25.1.1.1 ANALOG_MAPPING_QUERY

```
const int firmata::ANALOG_MAPPING_QUERY = 0x69  [static]
```

Definition at line 71 of file FirmataConstants.h.

Referenced by firmata::FirmataMarshaller::sendAnalogMappingQuery().

#### 25.1.1.2 ANALOG_MAPPING_RESPONSE

```
const int firmata::ANALOG_MAPPING_RESPONSE = 0x6A  [static]
```

Definition at line 72 of file FirmataConstants.h.

#### 25.1.1.3 ANALOG_MESSAGE

```
const int firmata::ANALOG_MESSAGE = 0xE0  [static]
```

Definition at line 39 of file FirmataConstants.h.

Referenced by firmata::FirmataParser::attach(), firmata::FirmataClass::FirmataClass(), firmata::FirmataParser↩ ::parse(), and firmata::FirmataMarshaller::sendAnalog().

#### 25.1.1.4 CAPABILITY_QUERY

```
const int firmata::CAPABILITY_QUERY = 0x6B  [static]
```

Definition at line 69 of file FirmataConstants.h.

Referenced by firmata::FirmataMarshaller::sendCapabilityQuery().

#### 25.1.1.5 CAPABILITY_RESPONSE

```
const int firmata::CAPABILITY_RESPONSE = 0x6C  [static]
```

Definition at line 70 of file FirmataConstants.h.

#### 25.1.1.6 DIGITAL_MESSAGE

```
const int firmata::DIGITAL_MESSAGE = 0x90  [static]
```

Definition at line 38 of file FirmataConstants.h.

Referenced by firmata::FirmataParser::attach(), firmata::FirmataClass::FirmataClass(), firmata::FirmataParser↩ ::parse(), and firmata::FirmataMarshaller::sendDigitalPort().

#### 25.1.1.7 ENCODER_DATA

```
const int firmata::ENCODER_DATA = 0x61  [static]
```

Definition at line 56 of file FirmataConstants.h.

#### 25.1.1.8 END_SYSEX

```
const int firmata::END_SYSEX = 0xF7  [static]
```

Definition at line 50 of file FirmataConstants.h.

Referenced by firmata::FirmataClass::endSysex(), firmata::FirmataParser::parse(), firmata::FirmataMarshaller↩ ::queryFirmwareVersion(), firmata::FirmataMarshaller::sendFirmwareVersion(), firmata::FirmataMarshaller::send↩ PinStateQuery(), and firmata::FirmataMarshaller::sendSysex().

#### 25.1.1.9 EXTENDED_ANALOG

```
const int firmata::EXTENDED_ANALOG = 0x6F  [static]
```

Definition at line 66 of file FirmataConstants.h.

#### 25.1.1.10 FIRMWARE_BUGFIX_VERSION

```
const int firmata::FIRMWARE_BUGFIX_VERSION = 7  [static]
```

Definition at line 24 of file FirmataConstants.h.

#### 25.1.1.11 FIRMWARE_MAJOR_VERSION

```
const int firmata::FIRMWARE_MAJOR_VERSION = 2  [static]
```

Definition at line 22 of file FirmataConstants.h.

#### 25.1.1.12 FIRMWARE_MINOR_VERSION

```
const int firmata::FIRMWARE_MINOR_VERSION = 5  [static]
```

Definition at line 23 of file FirmataConstants.h.

#### 25.1.1.13 I2C_CONFIG

```
const int firmata::I2C_CONFIG = 0x78  [static]
```

Definition at line 64 of file FirmataConstants.h.

#### 25.1.1.14 I2C_REPLY

```
const int firmata::I2C_REPLY = 0x77  [static]
```

Definition at line 63 of file FirmataConstants.h.

#### 25.1.1.15 I2C_REQUEST

```
const int firmata::I2C_REQUEST = 0x76  [static]
```

Definition at line 62 of file FirmataConstants.h.

#### 25.1.1.16 MAX_DATA_BYTES

```
const int firmata::MAX_DATA_BYTES = 64  [static]
```

Definition at line 34 of file FirmataConstants.h.

#### 25.1.1.17 ONEWIRE_DATA

```
const int firmata::ONEWIRE_DATA = 0x73  [static]
```

Definition at line 60 of file FirmataConstants.h.

#### 25.1.1.18 PIN_MODE_ANALOG

```
const int firmata::PIN_MODE_ANALOG = 0x02  [static]
```

Definition at line 81 of file FirmataConstants.h.

#### 25.1.1.19 PIN_MODE_ENCODER

```
const int firmata::PIN_MODE_ENCODER = 0x09  [static]
```

Definition at line 88 of file FirmataConstants.h.

#### 25.1.1.20 PIN_MODE_I2C

```
const int firmata::PIN_MODE_I2C = 0x06  [static]
```

Definition at line 85 of file FirmataConstants.h.

#### 25.1.1.21 PIN_MODE_IGNORE

```
const int firmata::PIN_MODE_IGNORE = 0x7F  [static]
```

Definition at line 91 of file FirmataConstants.h.

Referenced by firmata::FirmataClass::setPinMode().

#### 25.1.1.22 PIN_MODE_INPUT

```
const int firmata::PIN_MODE_INPUT = 0x00  [static]
```

Definition at line 79 of file FirmataConstants.h.

#### 25.1.1.23 PIN_MODE_ONEWIRE

```
const int firmata::PIN_MODE_ONEWIRE = 0x07  [static]
```

Definition at line 86 of file FirmataConstants.h.

#### 25.1.1.24 PIN_MODE_OUTPUT

```
const int firmata::PIN_MODE_OUTPUT = 0x01  [static]
```

Definition at line 80 of file FirmataConstants.h.

#### 25.1.1.25 PIN_MODE_PULLUP

```
const int firmata::PIN_MODE_PULLUP = 0x0B  [static]
```

Definition at line 90 of file FirmataConstants.h.

#### 25.1.1.26 PIN_MODE_PWM

```
const int firmata::PIN_MODE_PWM = 0x03  [static]
```

Definition at line 82 of file FirmataConstants.h.

#### 25.1.1.27 PIN_MODE_SERIAL

```
const int firmata::PIN_MODE_SERIAL = 0x0A  [static]
```

Definition at line 89 of file FirmataConstants.h.

#### 25.1.1.28 PIN_MODE_SERVO

```
const int firmata::PIN_MODE_SERVO = 0x04  [static]
```

Definition at line 83 of file FirmataConstants.h.

#### 25.1.1.29 PIN_MODE_SHIFT

```
const int firmata::PIN_MODE_SHIFT = 0x05  [static]
```

Definition at line 84 of file FirmataConstants.h.

#### 25.1.1.30 PIN_MODE_STEPPER

```
const int firmata::PIN_MODE_STEPPER = 0x08  [static]
```

Definition at line 87 of file FirmataConstants.h.

#### 25.1.1.31 PIN_STATE_QUERY

```
const int firmata::PIN_STATE_QUERY = 0x6D  [static]
```

Definition at line 67 of file FirmataConstants.h.

Referenced by firmata::FirmataMarshaller::sendPinStateQuery().

#### 25.1.1.32 PIN_STATE_RESPONSE

```
const int firmata::PIN_STATE_RESPONSE = 0x6E  [static]
```

Definition at line 68 of file FirmataConstants.h.

#### 25.1.1.33 PROTOCOL_BUGFIX_VERSION

```
const int firmata::PROTOCOL_BUGFIX_VERSION = 1  [static]
```

Definition at line 32 of file FirmataConstants.h.

#### 25.1.1.34 PROTOCOL_MAJOR_VERSION

```
const int firmata::PROTOCOL_MAJOR_VERSION = 2  [static]
```

Definition at line 30 of file FirmataConstants.h.

#### 25.1.1.35 PROTOCOL_MINOR_VERSION

```
const int firmata::PROTOCOL_MINOR_VERSION = 5  [static]
```

Definition at line 31 of file FirmataConstants.h.

#### 25.1.1.36 REPORT_ANALOG

```
const int firmata::REPORT_ANALOG = 0xC0  [static]
```

Definition at line 40 of file FirmataConstants.h.

Referenced by firmata::FirmataParser::attach(), firmata::FirmataClass::FirmataClass(), and firmata::Firmata↵ Parser::parse().

#### 25.1.1.37 REPORT_DIGITAL

```
const int firmata::REPORT_DIGITAL = 0xD0  [static]
```

Definition at line 41 of file FirmataConstants.h.

Referenced by firmata::FirmataParser::attach(), firmata::FirmataClass::FirmataClass(), and firmata::Firmata↵ Parser::parse().

#### 25.1.1.38 REPORT_FIRMWARE

```
const int firmata::REPORT_FIRMWARE = 0x79  [static]
```

Definition at line 65 of file FirmataConstants.h.

Referenced by firmata::FirmataParser::attach(), firmata::FirmataParser::detach(), firmata::FirmataClass::Firmata↩ Class(), firmata::FirmataMarshaller::queryFirmwareVersion(), and firmata::FirmataMarshaller::sendFirmware↩ Version().

#### 25.1.1.39 REPORT_VERSION

```
const int firmata::REPORT_VERSION = 0xF9  [static]
```

Definition at line 46 of file FirmataConstants.h.

Referenced by firmata::FirmataParser::attach(), firmata::FirmataParser::detach(), firmata::FirmataClass::Firmata↩ Class(), firmata::FirmataParser::parse(), firmata::FirmataMarshaller::queryVersion(), and firmata::Firmata↩ Marshaller::sendVersion().

#### 25.1.1.40 SAMPLING_INTERVAL

```
const int firmata::SAMPLING_INTERVAL = 0x7A  [static]
```

Definition at line 73 of file FirmataConstants.h.

Referenced by firmata::FirmataMarshaller::setSamplingInterval().

#### 25.1.1.41 SCHEDULER_DATA

```
const int firmata::SCHEDULER_DATA = 0x7B  [static]
```

Definition at line 74 of file FirmataConstants.h.

#### 25.1.1.42 SERIAL_DATA

```
const int firmata::SERIAL_DATA = 0x60  [static]
```

Definition at line 55 of file FirmataConstants.h.

#### 25.1.1.43 SERVO_CONFIG

```
const int firmata::SERVO_CONFIG = 0x70  [static]
```

Definition at line 57 of file FirmataConstants.h.

#### 25.1.1.44 SET_DIGITAL_PIN_VALUE

```
const int firmata::SET_DIGITAL_PIN_VALUE = 0xF5  [static]
```

Definition at line 44 of file FirmataConstants.h.

Referenced by firmata::FirmataParser::attach(), firmata::FirmataClass::FirmataClass(), firmata::FirmataParser↩ ::parse(), and firmata::FirmataMarshaller::sendDigital().

#### 25.1.1.45 SET_PIN_MODE

```
const int firmata::SET_PIN_MODE = 0xF4  [static]
```

Definition at line 43 of file FirmataConstants.h.

Referenced by firmata::FirmataParser::attach(), firmata::FirmataClass::FirmataClass(), firmata::FirmataParser↩ ::parse(), and firmata::FirmataMarshaller::sendPinMode().

#### 25.1.1.46 SHIFT_DATA

```
const int firmata::SHIFT_DATA = 0x75  [static]
```

Definition at line 61 of file FirmataConstants.h.

#### 25.1.1.47 START_SYSEX

```
const int firmata::START_SYSEX = 0xF0  [static]
```

Definition at line 49 of file FirmataConstants.h.

Referenced by firmata::FirmataParser::detach(), firmata::FirmataClass::detach(), firmata::FirmataClass::↩ FirmataClass(), firmata::FirmataParser::parse(), firmata::FirmataMarshaller::queryFirmwareVersion(), firmata↩ ::FirmataMarshaller::sendFirmwareVersion(), firmata::FirmataMarshaller::sendPinStateQuery(), firmata::Firmata↩ Marshaller::sendSysex(), and firmata::FirmataClass::startSysex().

#### 25.1.1.48 STEPPER_DATA

```
const int firmata::STEPPER_DATA = 0x72  [static]
```

Definition at line 59 of file FirmataConstants.h.

#### 25.1.1.49 STRING_DATA

```
const int firmata::STRING_DATA = 0x71  [static]
```

Definition at line 58 of file FirmataConstants.h.

Referenced by firmata::FirmataParser::attach(), firmata::FirmataClass::attach(), firmata::FirmataParser::detach(), firmata::FirmataClass::detach(), firmata::FirmataClass::FirmataClass(), firmata::FirmataMarshaller::sendString(), and firmata::FirmataClass::sendString().

#### 25.1.1.50 SYSEX_NON_REALTIME

```
const int firmata::SYSEX_NON_REALTIME = 0x7E  [static]
```

Definition at line 75 of file FirmataConstants.h.

#### 25.1.1.51 SYSEX_REALTIME

```
const int firmata::SYSEX_REALTIME = 0x7F  [static]
```

Definition at line 76 of file FirmataConstants.h.

#### 25.1.1.52 SYSTEM_RESET

```
const int firmata::SYSTEM_RESET = 0xFF  [static]
```

Definition at line 47 of file FirmataConstants.h.

Referenced by firmata::FirmataParser::attach(), firmata::FirmataClass::attach(), firmata::FirmataParser::detach(), firmata::FirmataClass::detach(), firmata::FirmataClass::FirmataClass(), firmata::FirmataParser::parse(), and firmata::FirmataMarshaller::systemReset().

#### 25.1.1.53 TOTAL_PIN_MODES

```
const int firmata::TOTAL_PIN_MODES = 13  [static]
```

Definition at line 93 of file FirmataConstants.h.

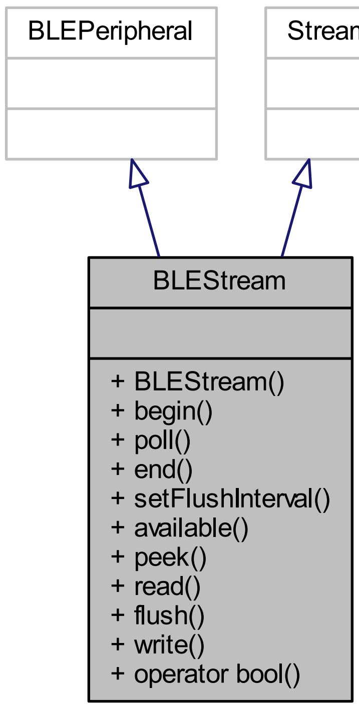
# Chapter 26

# Data Structure Documentation

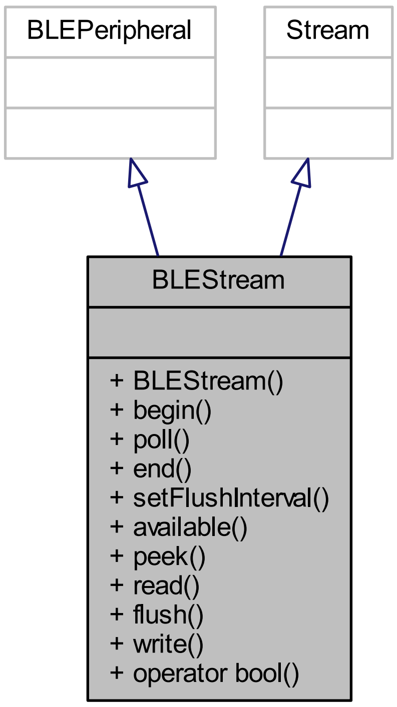
## 26.1 BLEStream Class Reference

```
#include <BLEStream.h>
```

Inheritance diagram for BLEStream:



Collaboration diagram for BLEStream:



### Public Member Functions

- BLEStream (unsigned char req=0, unsigned char rdy=0, unsigned char rst=0)
- void begin (...)
- bool poll ()
- void end ()
- void setFlushInterval (int)
- virtual int available (void)
- virtual int peek (void)
- virtual int read (void)
- virtual void flush (void)
- virtual size_t write (uint8_t byte)
- virtual operator bool ()

### 26.1.1 Detailed Description

Definition at line 27 of file BLEStream.h.

### 26.1.2 Constructor & Destructor Documentation

#### 26.1.2.1 BLEStream()

```
BLEStream::BLEStream (
            unsigned char req = 0,
            unsigned char rdy = 0,
            unsigned char rst = 0 )
```

Definition at line 78 of file BLEStream.h.

```
                                                                           :
#if defined(_VARIANT_ARDUINO_101_X_)
  BLEPeripheral()
#else
  BLEPeripheral(req, rdy, rst)
#endif
{
  this->_txCount = 0;
  this->_rxHead = this->_rxTail = 0;
  this->_flushed = 0;
  this->_flushInterval = BLESTREAM_TXBUFFER_FLUSH_INTERVAL;
  BLEStream::_instance = this;

  addAttribute(this->_uartService);
  addAttribute(this->_uartNameDescriptor);
  setAdvertisedServiceUuid(this->_uartService.uuid());
  addAttribute(this->_rxCharacteristic);
  addAttribute(this->_rxNameDescriptor);
  this->_rxCharacteristic.setEventHandler(BLEWritten, BLEStream::_received);
  addAttribute(this->_txCharacteristic);
  addAttribute(this->_txNameDescriptor);
}
```

References BLESTREAM_TXBUFFER_FLUSH_INTERVAL.

### 26.1.3 Member Function Documentation

#### 26.1.3.1 available()

```
int BLEStream::available (
            void  )  [virtual]
```

Definition at line 127 of file BLEStream.h.

```
{
// BLEPeripheral::poll only calls delay(1) in CurieBLE so skipping it here to avoid the delay
#ifndef _VARIANT_ARDUINO_101_X_
  // TODO Need to do more testing to determine if all of these calls to BLEPeripheral::poll are
  // actually necessary. Seems to run fine without them, but only minimal testing so far.
  BLEPeripheral::poll();
#endif
  int retval = (this->_rxHead - this->_rxTail + sizeof(this->_rxBuffer)) % sizeof(this->_rxBuffer);
#ifdef BLE_SERIAL_DEBUG
  if (retval > 0) {
    Serial.print(F("BLEStream::available() = "));
    Serial.println(retval);
  }
#endif
  return retval;
}
```

#### 26.1.3.2 begin()

```
void BLEStream::begin (
              ... )
```

Definition at line 101 of file BLEStream.h.

```
{
  BLEPeripheral::begin();
#ifdef BLE_SERIAL_DEBUG
  Serial.println(F("BLEStream::begin()"));
#endif
}
```

#### 26.1.3.3 end()

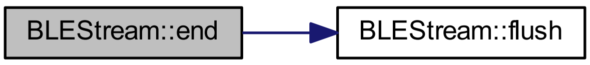
```
void BLEStream::end (
             void )
```

Definition at line 119 of file BLEStream.h.

```
{
  this->_rxCharacteristic.setEventHandler(BLEWritten, (void(*)(BLECentral&, BLECharacteristic&))NULL);
  this->_rxHead = this->_rxTail = 0;
  flush();
  BLEPeripheral::disconnect();
}
```

References flush().

Here is the call graph for this function:



#### 26.1.3.4 flush()

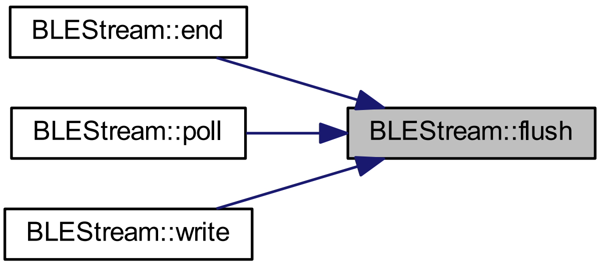
```
void BLEStream::flush (
             void ) [virtual]
```

Definition at line 174 of file BLEStream.h.

```
{
  if (this->_txCount == 0) return;
#ifndef _VARIANT_ARDUINO_101_X_
  // ensure there are available packets before sending
  while(!this->_txCharacteristic.canNotify()) {
    BLEPeripheral::poll();
  }
#endif
  this->_txCharacteristic.setValue(this->_txBuffer, this->_txCount);
  this->_flushed = millis();
  this->_txCount = 0;
#ifdef BLE_SERIAL_DEBUG
  Serial.println(F("BLEStream::flush()"));
#endif
}
```

Referenced by end(), poll(), and write().

Here is the caller graph for this function:



### 26.1.3.5 operator bool()

```
BLEStream::operator bool ( )  [virtual]
```

Definition at line 207 of file BLEStream.h.

```
{
  bool retval = this->_connected = BLEPeripheral::connected();
#ifdef BLE_SERIAL_DEBUG
  Serial.print(F("BLEStream::operator bool() = "));
  Serial.println(retval);
#endif
  return retval;
}
```

### 26.1.3.6 peek()

```
int BLEStream::peek (
              void  )  [virtual]
```

Definition at line 145 of file BLEStream.h.

```
{
#ifndef _VARIANT_ARDUINO_101_X_
  BLEPeripheral::poll();
#endif
  if (this->_rxTail == this->_rxHead) return -1;
  uint8_t byte = this->_rxBuffer[this->_rxTail];
#ifdef BLE_SERIAL_DEBUG
  Serial.print(F("BLEStream::peek() = 0x"));
  Serial.println(byte, HEX);
#endif
  return byte;
}
```

#### 26.1.3.7 poll()

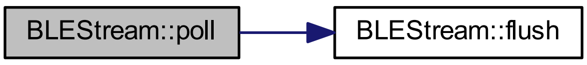
```
bool BLEStream::poll ( )
```

Definition at line 109 of file BLEStream.h.

```
{
  // BLEPeripheral::poll is called each time connected() is called
  this->_connected = BLEPeripheral::connected();
  if (millis() > this->_flushed + this->_flushInterval) {
    flush();
  }
  return this->_connected;
}
```

References flush().

Here is the call graph for this function:



#### 26.1.3.8 read()

```
int BLEStream::read (
              void  )  [virtual]
```

Definition at line 159 of file BLEStream.h.

```
{
#ifndef _VARIANT_ARDUINO_101_X_
  BLEPeripheral::poll();
#endif
  if (this->_rxTail == this->_rxHead) return -1;
  this->_rxTail = (this->_rxTail + 1) % sizeof(this->_rxBuffer);
  uint8_t byte = this->_rxBuffer[this->_rxTail];
#ifdef BLE_SERIAL_DEBUG
  Serial.print(F("BLEStream::read() = 0x"));
  Serial.println(byte, HEX);
#endif
  return byte;
}
```

#### 26.1.3.9 setFlushInterval()

```
void BLEStream::setFlushInterval (
              int interval )
```

Definition at line 217 of file BLEStream.h.

```
{
  if (interval > BLESTREAM_MIN_FLUSH_INTERVAL) {
    this->_flushInterval = interval;
  }
}
```

References BLESTREAM_MIN_FLUSH_INTERVAL.

#### 26.1.3.10 write()

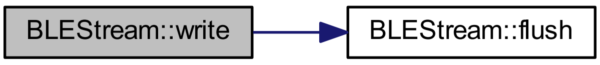
```
size_t BLEStream::write (
            uint8_t byte )  [virtual]
```

Definition at line 191 of file BLEStream.h.

```
{
#ifndef _VARIANT_ARDUINO_101_X_
  BLEPeripheral::poll();
#endif
  if (this->_txCharacteristic.subscribed() == false) return 0;
  this->_txBuffer[this->_txCount++] = byte;
  if (this->_txCount == sizeof(this->_txBuffer)) flush();
#ifdef BLE_SERIAL_DEBUG
  Serial.print(F("BLEStream::write( 0x"));
  Serial.print(byte, HEX);
  Serial.println(F(") = 1"));
#endif
  return 1;
}
```

References flush().

Here is the call graph for this function:



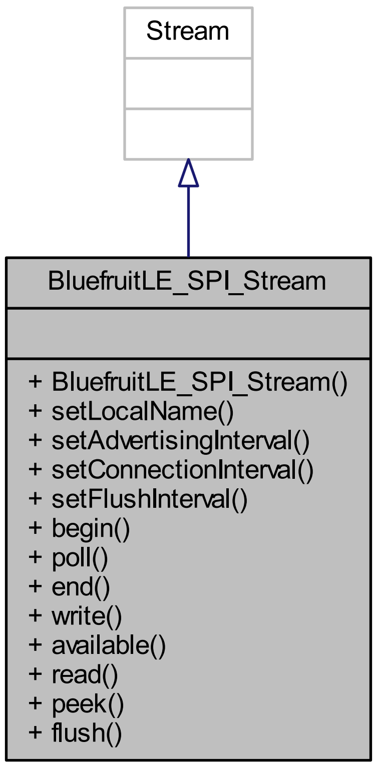
The documentation for this class was generated from the following file:

- BLEStream.h

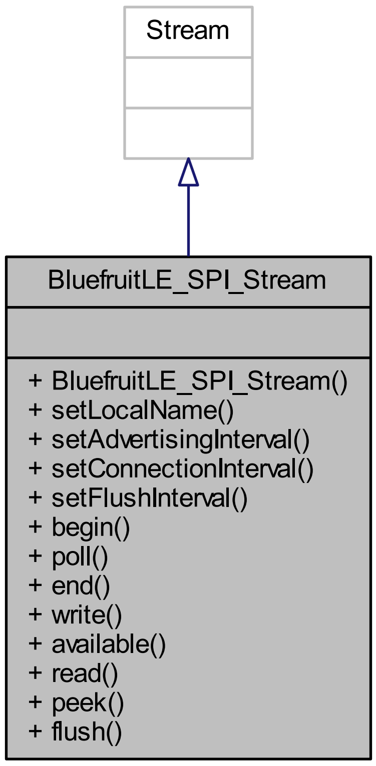
## 26.2 BluefruitLE_SPI_Stream Class Reference

```
#include <BluefruitLE_SPI_Stream.h>
```

Inheritance diagram for BluefruitLE_SPI_Stream:



Collaboration diagram for BluefruitLE_SPI_Stream:



## Public Member Functions

- BluefruitLE_SPI_Stream (int8_t csPin, int8_t irqPin, int8_t rstPin)
- void setLocalName (const char ∗localName)
- void setAdvertisingInterval (unsigned short advertisingInterval)
- void setConnectionInterval (unsigned short minConnInterval, unsigned short maxConnInterval)
- void setFlushInterval (int flushInterval)
- void begin ()
- bool poll ()
- void end ()
- size_t write (uint8_t byte)
- int available ()
- int read ()
- int peek ()
- void flush ()

### 26.2.1 Detailed Description

Definition at line 14 of file BluefruitLE_SPI_Stream.h.

## 26.2.2 Constructor & Destructor Documentation

### 26.2.2.1 BluefruitLE_SPI_Stream()

```
BluefruitLE_SPI_Stream::BluefruitLE_SPI_Stream (
             int8_t csPin,
             int8_t irqPin,
             int8_t rstPin )
```

Definition at line 51 of file BluefruitLE_SPI_Stream.h.

```
                                                                                    :
  ble(csPin, irqPin, rstPin),
  advertisingInterval(0),
  minConnInterval(0),
  maxConnInterval(0),
  txCount(0)
{ }
```

## 26.2.3 Member Function Documentation

### 26.2.3.1 available()

```
int BluefruitLE_SPI_Stream::available (
             void  )
```

Definition at line 145 of file BluefruitLE_SPI_Stream.h.

```
{
  return ble.available();
}
```

### 26.2.3.2 begin()

```
void BluefruitLE_SPI_Stream::begin (
             void  )
```

Definition at line 80 of file BluefruitLE_SPI_Stream.h.

```
{
  // Initialize the SPI interface
  ble.begin();

  // Perform a factory reset to make sure everything is in a known state
  ble.factoryReset();

  // Disable command echo from Bluefruit
  ble.echo(false);

  // Change the MODE LED to indicate BLE UART activity
  ble.println("AT+HWMODELED=BLEUART");

  // Set local name
  if (localName.length() > 0) {
    ble.print("AT+GAPDEVNAME=");
    ble.println(localName);
  }

  // Set connection and advertising intervals
  ble.print("AT+GAPINTERVALS=");
  if (minConnInterval > 0) ble.print(minConnInterval);
  ble.print(",");
  if (maxConnInterval > 0) ble.print(maxConnInterval);
  ble.print(",");
  if (advertisingInterval > 0) ble.print(advertisingInterval);
  ble.print(",,");  // Always omit fast advertising timeout, hence two commas
  if (advertisingInterval > 0) ble.print(advertisingInterval);
  ble.println();

  // Disable real and simulated mode switch (i.e. "+++") command
  ble.println("AT+MODESWITCHEN=local,0");
  ble.enableModeSwitchCommand(false);

  // Switch to data mode
  ble.setMode(BLUEFRUIT_MODE_DATA);
}
```

#### 26.2.3.3 end()

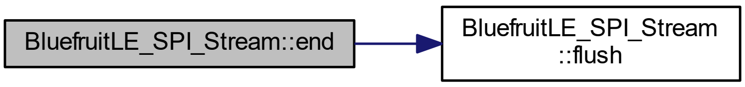
```
void BluefruitLE_SPI_Stream::end (
              void  )
```

Definition at line 132 of file BluefruitLE_SPI_Stream.h.

```
{
  flush();
  ble.end();
}
```

References flush().

Here is the call graph for this function:



#### 26.2.3.4 flush()

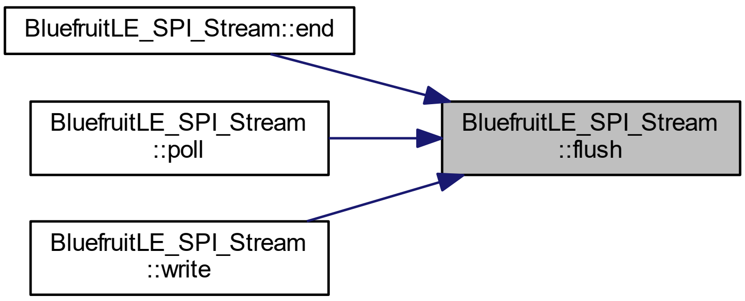
```
void BluefruitLE_SPI_Stream::flush (
              void  )
```

Definition at line 160 of file BluefruitLE_SPI_Stream.h.

```
{
  ble.write(txBuffer, txCount);
  txCount = 0;
}
```

Referenced by end(), poll(), and write().

Here is the caller graph for this function:


BluefruitLE_SPI_Stream::end
BluefruitLE_SPI_Stream ::poll
BluefruitLE_SPI_Stream ::write
BluefruitLE_SPI_Stream ::flush

#### 26.2.3.5 peek()

```
int BluefruitLE_SPI_Stream::peek (
             void  )
```

Definition at line 155 of file BluefruitLE_SPI_Stream.h.

```
{
  return ble.peek();
}
```

#### 26.2.3.6 poll()

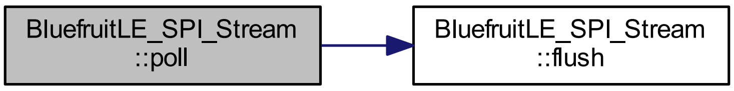
```
bool BluefruitLE_SPI_Stream::poll ( )
```

Definition at line 119 of file BluefruitLE_SPI_Stream.h.

```
{
  // If there's outgoing data in the buffer, just send it.  The firmware on
  // the nRF51822 will decide when to transmit the data in its TX FIFO.
  if (txCount) flush();

  // In order to check for a connection, we would need to switch from data to
  // command mode and back again.  However, due to the internal workings of
  // Adafruit_BluefruitLE_SPI, this can lead to unread incoming data being
  // lost.  Therefore, we always return true.
  return true;
}
```

References flush().

Here is the call graph for this function:



#### 26.2.3.7 read()

```
int BluefruitLE_SPI_Stream::read (
            void )
```

Definition at line 150 of file BluefruitLE_SPI_Stream.h.

```
{
  return ble.read();
}
```

#### 26.2.3.8 setAdvertisingInterval()

```
void BluefruitLE_SPI_Stream::setAdvertisingInterval (
            unsigned short advertisingInterval )
```

Definition at line 64 of file BluefruitLE_SPI_Stream.h.

```
{
  this->advertisingInterval = advertisingInterval;
}
```

#### 26.2.3.9 setConnectionInterval()

```
void BluefruitLE_SPI_Stream::setConnectionInterval (
            unsigned short minConnInterval,
            unsigned short maxConnInterval )
```

Definition at line 69 of file BluefruitLE_SPI_Stream.h.

```
{
  this->minConnInterval = minConnInterval;
  this->maxConnInterval = maxConnInterval;
}
```

#### 26.2.3.10 setFlushInterval()

```
void BluefruitLE_SPI_Stream::setFlushInterval (
            int flushInterval )
```

Definition at line 75 of file BluefruitLE_SPI_Stream.h.

```
{
  // Not used
}
```

#### 26.2.3.11 setLocalName()

```
void BluefruitLE_SPI_Stream::setLocalName (
             const char * localName )
```

Definition at line 59 of file BluefruitLE_SPI_Stream.h.

```
{
  this->localName = localName;
}
```

#### 26.2.3.12 write()

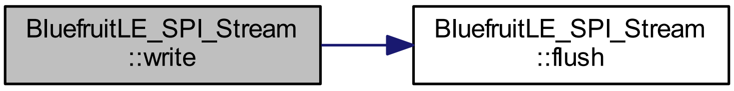
```
size_t BluefruitLE_SPI_Stream::write (
             uint8_t byte )
```

Definition at line 138 of file BluefruitLE_SPI_Stream.h.

```
{
  txBuffer[txCount++] = byte;
  if (txCount == sizeof(txBuffer)) flush();
  return 1;
}
```

References flush().

Here is the call graph for this function:


BluefruitLE_SPI_Stream ::write
BluefruitLE_SPI_Stream ::flush

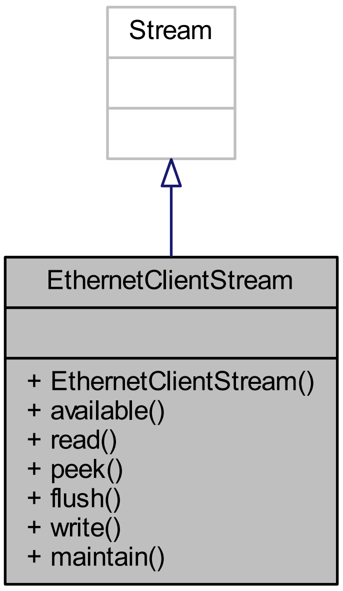
The documentation for this class was generated from the following file:

- BluefruitLE_SPI_Stream.h

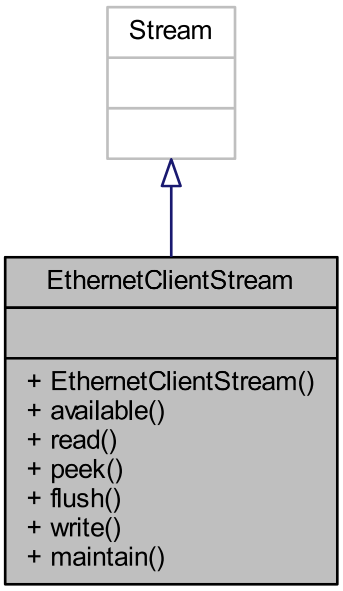
## 26.3 EthernetClientStream Class Reference

```
#include <EthernetClientStream.h>
```

Inheritance diagram for EthernetClientStream:



Collaboration diagram for EthernetClientStream:



## Public Member Functions

- EthernetClientStream (Client &client, IPAddress localip, IPAddress ip, const char ∗host, uint16_t port)
- int available ()
- int read ()
- int peek ()
- void flush ()
- size_t write (uint8_t)
- void maintain (IPAddress localip)

### 26.3.1 Detailed Description

Definition at line 31 of file EthernetClientStream.h.

### 26.3.2 Constructor & Destructor Documentation

#### 26.3.2.1 EthernetClientStream()

```
EthernetClientStream::EthernetClientStream (
            Client & client,
            IPAddress localip,
            IPAddress ip,
            const char * host,
            uint16_t port )
```

Definition at line 60 of file EthernetClientStream.h.

```
61   : client(client),
62     localip(localip),
63     ip(ip),
64     host(host),
65     port(port),
66     connected(false)
67 {
68 }
```

### 26.3.3 Member Function Documentation

#### 26.3.3.1 available()

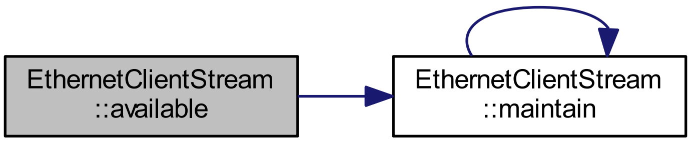
```
int EthernetClientStream::available (
            void )
```

Definition at line 71 of file EthernetClientStream.h.

```
{
  return maintain() ? client.available() : 0;
}
```

References maintain().

Here is the call graph for this function:



#### 26.3.3.2 flush()

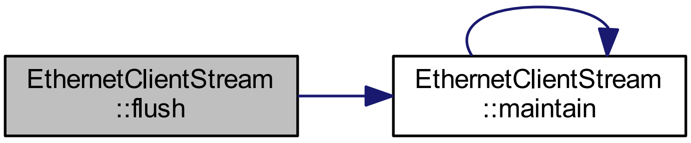
```
void EthernetClientStream::flush (
            void )
```

Definition at line 88 of file EthernetClientStream.h.

```
{
  if (maintain())
    client.flush();
}
```

References maintain().

Here is the call graph for this function:



#### 26.3.3.3 maintain()

```
void EthernetClientStream::maintain (
              IPAddress localip )
```

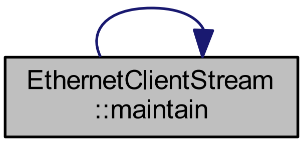
Definition at line 101 of file EthernetClientStream.h.

```
{
  // ensure the local IP is updated in the case that it is changed by the DHCP server
  if (this->localip != localip) {
    this->localip = localip;
    if (connected)
      stop();
  }
}
```

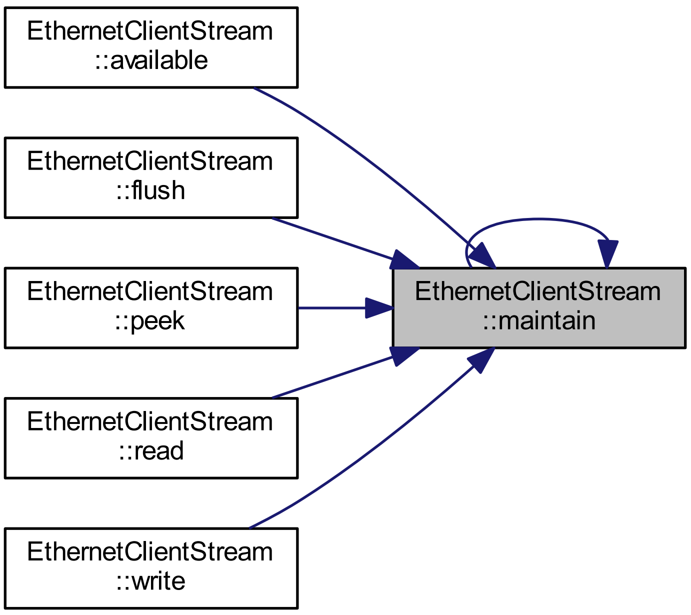
References maintain().

Referenced by available(), flush(), maintain(), peek(), read(), and write().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 26.3.3.4 peek()

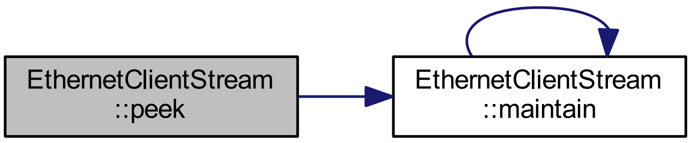
```
int EthernetClientStream::peek (
              void  )
```

Definition at line 83 of file EthernetClientStream.h.

```
{
  return maintain() ? client.peek() : -1;
}
```

References maintain().

Here is the call graph for this function:



#### 26.3.3.5 read()

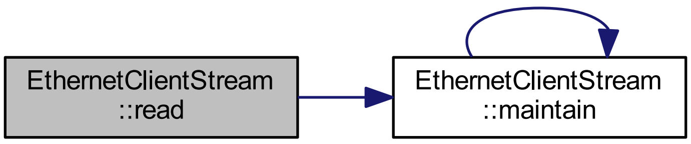
```
int EthernetClientStream::read (
              void  )
```

Definition at line 77 of file EthernetClientStream.h.

```
{
  return maintain() ? client.read() : -1;
}
```

References maintain().

Here is the call graph for this function:



#### 26.3.3.6 write()

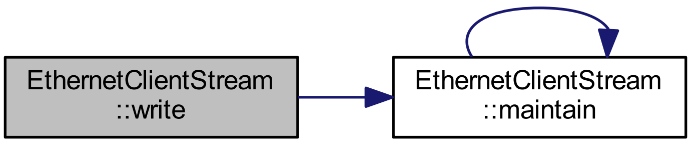
```
size_t EthernetClientStream::write (
            uint8_t c )
```

Definition at line 95 of file EthernetClientStream.h.

```
96 {
97   return maintain() ? client.write(c) : 0;
98 }
```

References maintain().

Here is the call graph for this function:



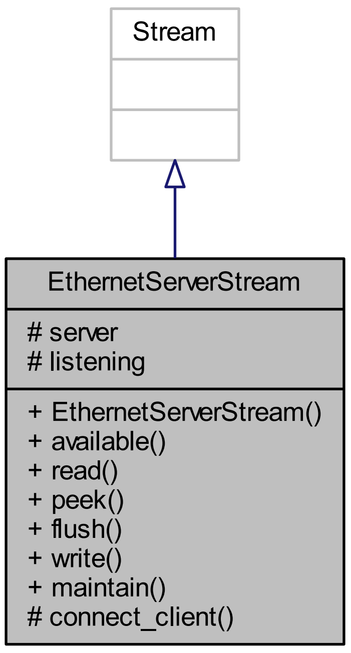
The documentation for this class was generated from the following file:

- EthernetClientStream.h

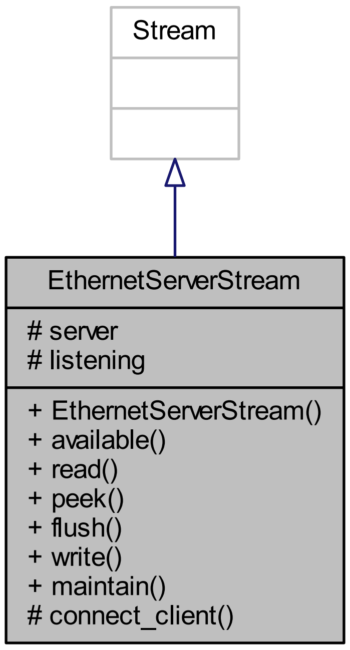
## 26.4 EthernetServerStream Class Reference

```
#include <EthernetServerStream.h>
```

Inheritance diagram for EthernetServerStream:



Collaboration diagram for EthernetServerStream:



## Public Member Functions

- EthernetServerStream (IPAddress localip, uint16_t port)
- int available ()
- int read ()
- int peek ()
- void flush ()
- size_t write (uint8_t)
- void maintain (IPAddress localip)

## Protected Member Functions

- bool connect_client ()

## Protected Attributes

- EthernetServer server = EthernetServer(3030)
- bool listening = false

### 26.4.1 Detailed Description

Definition at line 26 of file EthernetServerStream.h.

## 26.4.2 Constructor & Destructor Documentation

#### 26.4.2.1 EthernetServerStream()

```
EthernetServerStream::EthernetServerStream (
                IPAddress localip,
                uint16_t port )
```

Definition at line 57 of file EthernetServerStream.h.

```
  : localip(localip),
    port(port),
    connected(false)
{
}
```

## 26.4.3 Member Function Documentation

#### 26.4.3.1 available()

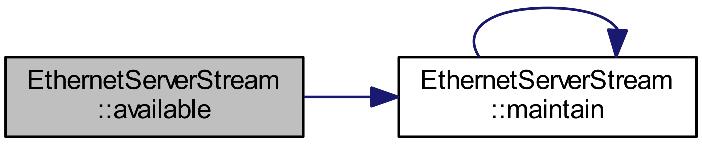
```
int EthernetServerStream::available (
                void )
```

Definition at line 81 of file EthernetServerStream.h.

```
{
  return maintain() ? client.available() : 0;
}
```

References maintain().

Here is the call graph for this function:


EthernetServerStream ::available
EthernetServerStream ::maintain

#### 26.4.3.2 connect_client()

```
bool EthernetServerStream::connect_client ( )  [protected]
```

Definition at line 64 of file EthernetServerStream.h.

```
{
  if ( connected )
  {
    if ( client && client.connected() ) return true;
    stop();
  }

  EthernetClient newClient = server.available();
  if ( !newClient ) return false;
  client = newClient;
  connected = true;
  DEBUG_PRINTLN("Connected");
  return true;
}
```

References DEBUG_PRINTLN, and server.

#### 26.4.3.3 flush()

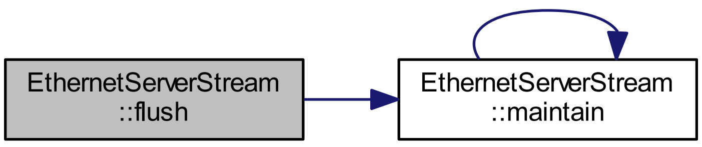
```
void EthernetServerStream::flush (
            void  )
```

Definition at line 98 of file EthernetServerStream.h.

```
{
  if (maintain())
    client.flush();
}
```

References maintain().

Here is the call graph for this function:



#### 26.4.3.4 maintain()

```
void EthernetServerStream::maintain (
              IPAddress localip )
```

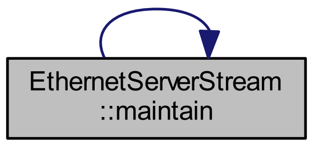
Definition at line 111 of file EthernetServerStream.h.

```
{
  // ensure the local IP is updated in the case that it is changed by the DHCP server
  if (this->localip != localip) {
    this->localip = localip;
    if (connected)
      stop();
  }
}
```

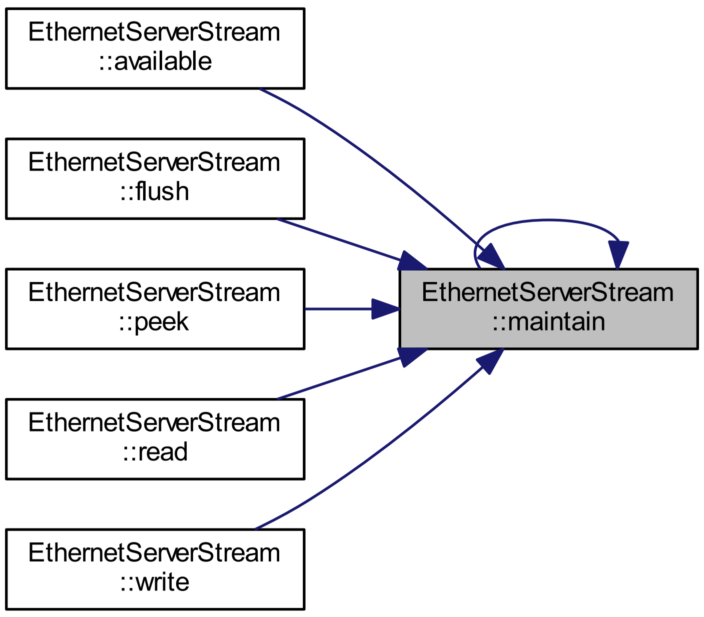
References maintain().

Referenced by available(), flush(), maintain(), peek(), read(), and write().

Here is the call graph for this function:



Here is the caller graph for this function:



### 26.4.3.5 peek()

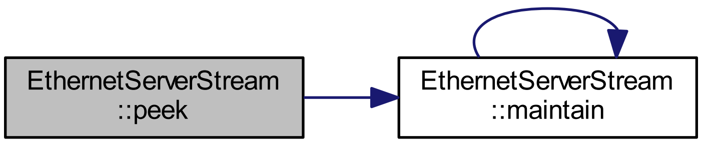
```
int EthernetServerStream::peek (
             void  )
```

Definition at line 93 of file EthernetServerStream.h.

```
{
  return maintain() ? client.peek() : -1;
}
```

References maintain().

Here is the call graph for this function:



### 26.4.3.6 read()

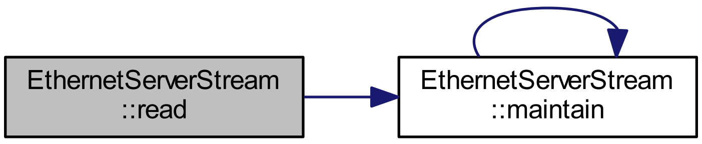
```
int EthernetServerStream::read (
             void  )
```

Definition at line 87 of file EthernetServerStream.h.

```
{
  return maintain() ? client.read() : -1;
}
```

References maintain().

Here is the call graph for this function:



#### 26.4.3.7 write()

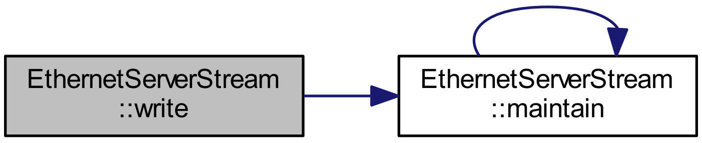
```
size_t EthernetServerStream::write (
             uint8_t c )
```

Definition at line 105 of file EthernetServerStream.h.

```
{
  return maintain() ? client.write(c) : 0;
}
```

References maintain().

Here is the call graph for this function:



### 26.4.4 Field Documentation

#### 26.4.4.1 listening

```
bool EthernetServerStream::listening = false  [protected]
```

Definition at line 47 of file EthernetServerStream.h.

#### 26.4.4.2 server

```
EthernetServer EthernetServerStream::server = EthernetServer(3030)  [protected]
```

Definition at line 46 of file EthernetServerStream.h.

Referenced by connect_client().

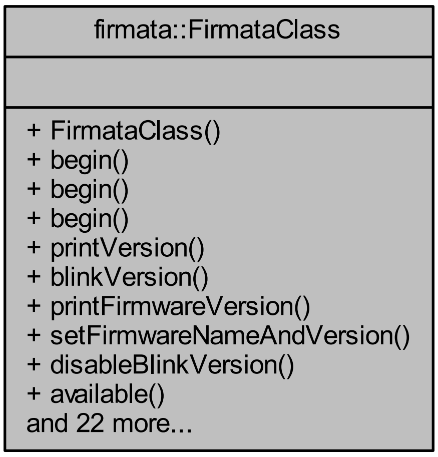
The documentation for this class was generated from the following file:

- EthernetServerStream.h

## 26.5 firmata::FirmataClass Class Reference

```
#include <Firmata.h>
```

Collaboration diagram for firmata::FirmataClass:



### Public Types

- typedef void(∗ callbackFunction) (uint8_t, int)
- typedef void(∗ systemCallbackFunction) (void)
- typedef void(∗ stringCallbackFunction) (char ∗)
- typedef void(∗ sysexCallbackFunction) (uint8_t command, uint8_t argc, uint8_t ∗argv)

### Public Member Functions

- FirmataClass ()
- void begin ()
- void begin (long)
- void begin (Stream &s)
- void printVersion (void)
- void blinkVersion (void)
- void printFirmwareVersion (void)
- void setFirmwareNameAndVersion (const char ∗name, byte major, byte minor)
- void disableBlinkVersion ()
- int available (void)
- void processInput (void)
- void parse (unsigned char value)
- boolean isParsingMessage (void)
- void sendAnalog (byte pin, int value)
- void sendDigital (byte pin, int value)
- void sendDigitalPort (byte portNumber, int portData)

- void sendString (const char ∗string)
- void sendString (byte command, const char ∗string)
- void sendSysex (byte command, byte bytec, byte ∗bytev)
- void write (byte c)
- void attach (uint8_t command, callbackFunction newFunction)
- void attach (uint8_t command, systemCallbackFunction newFunction)
- void attach (uint8_t command, stringCallbackFunction newFunction)
- void attach (uint8_t command, sysexCallbackFunction newFunction)
- void detach (uint8_t command)
- byte getPinMode (byte pin)
- void setPinMode (byte pin, byte config)
- int getPinState (byte pin)
- void setPinState (byte pin, int state)
- void sendValueAsTwo7bitBytes (int value)
- void startSysex (void)
- void endSysex (void)

### Friends

- void FirmataMarshaller::encodeByteStream (size_t bytec, uint8_t ∗bytev, size_t max_bytes) const

## 26.5.1 Detailed Description

Definition at line 54 of file Firmata.h.

## 26.5.2 Member Typedef Documentation

### 26.5.2.1 callbackFunction

```
typedef void(∗ firmata::FirmataClass::callbackFunction) (uint8_t, int)
```

Definition at line 57 of file Firmata.h.

### 26.5.2.2 stringCallbackFunction

```
typedef void(∗ firmata::FirmataClass::stringCallbackFunction) (char ∗)
```

Definition at line 59 of file Firmata.h.

#### 26.5.2.3 sysexCallbackFunction

`typedef void(* firmata::FirmataClass::sysexCallbackFunction) (uint8_t command, uint8_t argc, uint8_t *argv)`

Definition at line 60 of file Firmata.h.

#### 26.5.2.4 systemCallbackFunction

`typedef void(* firmata::FirmataClass::systemCallbackFunction) (void)`

Definition at line 58 of file Firmata.h.

### 26.5.3 Constructor & Destructor Documentation

#### 26.5.3.1 FirmataClass()

`FirmataClass::FirmataClass ( )`

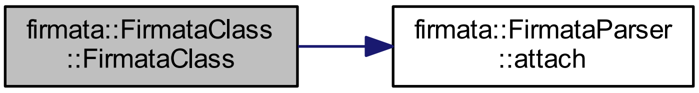
The Firmata class. An instance named "Firmata" is created automatically for the user.

Definition at line 80 of file Firmata.cpp.

```
:
  parser(FirmataParser(parserBuffer, MAX_DATA_BYTES))
{
  firmwareVersionCount = 0;
  firmwareVersionVector = 0;
  blinkVersionDisabled = false;

  // Establish callback translation to parser callbacks
  parser.attach(ANALOG_MESSAGE, (FirmataParser::callbackFunction)staticAnalogCallback, (void *)NULL);
  parser.attach(DIGITAL_MESSAGE, (FirmataParser::callbackFunction)staticDigitalCallback, (void *)NULL);
  parser.attach(REPORT_ANALOG, (FirmataParser::callbackFunction)staticReportAnalogCallback, (void
    *)NULL);
  parser.attach(REPORT_DIGITAL, (FirmataParser::callbackFunction)staticReportDigitalCallback, (void
    *)NULL);
  parser.attach(SET_PIN_MODE, (FirmataParser::callbackFunction)staticPinModeCallback, (void *)NULL);
  parser.attach(SET_DIGITAL_PIN_VALUE, (FirmataParser::callbackFunction)staticPinValueCallback, (void
    *)NULL);
  parser.attach(STRING_DATA, (FirmataParser::stringCallbackFunction)staticStringCallback, (void *)NULL);
  parser.attach(START_SYSEX, (FirmataParser::sysexCallbackFunction)staticSysexCallback, (void *)NULL);
  parser.attach(REPORT_FIRMWARE, (FirmataParser::versionCallbackFunction)staticReportFirmwareCallback,
    this);
  parser.attach(REPORT_VERSION, (FirmataParser::systemCallbackFunction)staticReportVersionCallback,
    this);
  parser.attach(SYSTEM_RESET, (FirmataParser::systemCallbackFunction)staticSystemResetCallback, (void
    *)NULL);
}
```

References firmata::ANALOG_MESSAGE, firmata::FirmataParser::attach(), firmata::DIGITAL_MESSAGE, firmata::REPORT_ANALOG, firmata::REPORT_DIGITAL, firmata::REPORT_FIRMWARE, firmata::REPORT↩_VERSION, firmata::SET_DIGITAL_PIN_VALUE, firmata::SET_PIN_MODE, firmata::START_SYSEX, firmata::↩STRING_DATA, and firmata::SYSTEM_RESET.

Here is the call graph for this function:


firmata::FirmataClass ::FirmataClass
firmata::FirmataParser ::attach

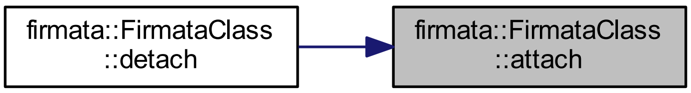
### 26.5.4 Member Function Documentation

#### 26.5.4.1 attach() [1/4]

```
void firmata::FirmataClass::attach (
            uint8_t command,
            callbackFunction newFunction )
```

Referenced by detach().

Here is the caller graph for this function:


firmata::FirmataClass ::detach
firmata::FirmataClass ::attach

#### 26.5.4.2 attach() [2/4]

```
void FirmataClass::attach (
            uint8_t command,
            stringCallbackFunction newFunction )
```

Attach a callback function for the STRING_DATA command.

**Parameters**

| | |
|---|---|
| *command* | Must be set to STRING_DATA or it will be ignored. |
| *newFunction* | A reference to the string callback function to attach. |

Definition at line 427 of file Firmata.cpp.

```
{
  switch (command) {
    case STRING_DATA:
      currentStringCallback = newFunction;
      break;
  }
}
```

References firmata::STRING_DATA.

#### 26.5.4.3 attach() [3/4]

```
void FirmataClass::attach (
            uint8_t command,
            sysexCallbackFunction newFunction )
```

Attach a generic sysex callback function to sysex command.

**Parameters**

| | |
|---|---|
| *command* | The ID of the command to attach a callback function to. |
| *newFunction* | A reference to the sysex callback function to attach. |

Definition at line 441 of file Firmata.cpp.

```
{
  (void)command;
  currentSysexCallback = newFunction;
}
```

#### 26.5.4.4 attach() [4/4]

```
void FirmataClass::attach (
            uint8_t command,
            systemCallbackFunction newFunction )
```

Attach a callback function for the SYSTEM_RESET command.

**Parameters**

| | |
|---|---|
| *command* | Must be set to SYSTEM_RESET or it will be ignored. |
| *newFunction* | A reference to the system reset callback function to attach. |

Definition at line 413 of file Firmata.cpp.

```
{
  switch (command) {
    case SYSTEM_RESET:
      currentSystemResetCallback = newFunction;
      break;
  }
}
```

References firmata::SYSTEM_RESET.

#### 26.5.4.5 available()

```
int FirmataClass::available (
            void  )
```

A wrapper for Stream::available()

**Returns**

The number of bytes remaining in the input stream buffer.

Definition at line 244 of file Firmata.cpp.

```
{
  return FirmataStream->available();
}
```

#### 26.5.4.6 begin() [1/3]

```
void FirmataClass::begin (
            void   )
```

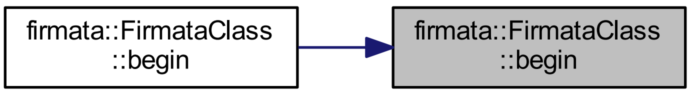
Initialize the default Serial transport at the default baud of 57600.

Definition at line 109 of file Firmata.cpp.

```
{
  begin(57600);
}
```

Referenced by begin().

Here is the caller graph for this function:



#### 26.5.4.7 begin() [2/3]

```
void FirmataClass::begin (
            long speed )
```

Initialize the default Serial transport and override the default baud. Sends the protocol version to the host application followed by the firmware version and name. blinkVersion is also called. To skip the call to blinkVersion, call Firmata.disableBlinkVersion() before calling Firmata.begin(baud).

**Parameters**

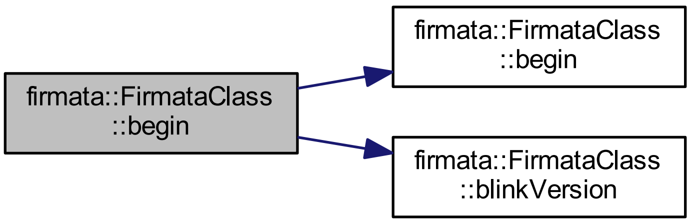
| | |
|---|---|
| *speed* | The baud to use. 57600 baud is the default value. |

Definition at line 121 of file Firmata.cpp.

```
{
  Serial.begin(speed);
  blinkVersion();
  begin(Serial);
}
```

References begin(), and blinkVersion().

Here is the call graph for this function:


firmata::FirmataClass ::begin
firmata::FirmataClass ::begin
firmata::FirmataClass ::blinkVersion

#### 26.5.4.8 begin() [3/3]

```
void FirmataClass::begin (
             Stream & s )
```

Reassign the Firmata stream transport.

**Parameters**

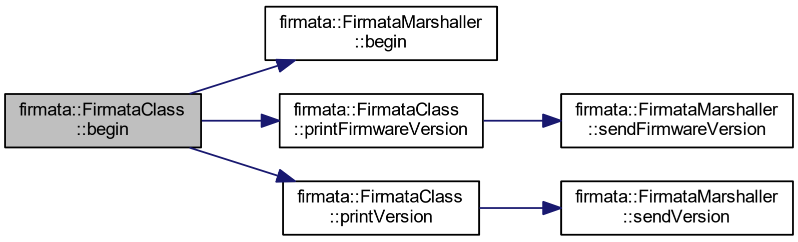
| | |
|---|---|
| *s* | A reference to the Stream transport object. This can be any type of transport that implements the Stream interface. Some examples include Ethernet, WiFi and other UARTs on the board (Serial1, Serial2, etc). |

Definition at line 134 of file Firmata.cpp.

```
{
  FirmataStream = &s;
  marshaller.begin(s);
  // do not call blinkVersion() here because some hardware such as the
  // Ethernet shield use pin 13
  printVersion();         // send the protocol version
  printFirmwareVersion(); // send the firmware name and version
}
```

References firmata::FirmataMarshaller::begin(), printFirmwareVersion(), and printVersion().

Here is the call graph for this function:


firmata::FirmataClass ::begin
firmata::FirmataMarshaller ::begin
firmata::FirmataClass ::printFirmwareVersion
firmata::FirmataMarshaller ::sendFirmwareVersion
firmata::FirmataClass ::printVersion
firmata::FirmataMarshaller ::sendVersion

### 26.5.4.9 blinkVersion()

```
void FirmataClass::blinkVersion (
            void )
```

Blink the Firmata protocol version to the onboard LEDs (if the board has an onboard LED). If VERSION_BLINK_PIN is not defined in Boards.h for a particular board, then this method does nothing. The first series of flashes indicates the firmware major version (2 flashes = 2). The second series of flashes indicates the firmware minor version (5 flashes = 5).

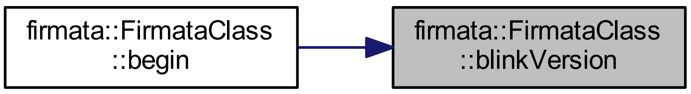
Definition at line 159 of file Firmata.cpp.

```
{
#if defined(VERSION_BLINK_PIN)
  if (blinkVersionDisabled) return;
  // flash the pin with the protocol version
  pinMode(VERSION_BLINK_PIN, OUTPUT);
  strobeBlinkPin(VERSION_BLINK_PIN, FIRMATA_FIRMWARE_MAJOR_VERSION, 40, 210);
  delay(250);
  strobeBlinkPin(VERSION_BLINK_PIN, FIRMATA_FIRMWARE_MINOR_VERSION, 40, 210);
  delay(125);
#endif
}
```

References FIRMATA_FIRMWARE_MAJOR_VERSION, and FIRMATA_FIRMWARE_MINOR_VERSION.

Referenced by begin().

Here is the caller graph for this function:


firmata::FirmataClass ::begin
firmata::FirmataClass ::blinkVersion

### 26.5.4.10 detach()

```
void FirmataClass::detach (
            uint8_t command )
```

Detach a callback function for a specified command (such as SYSTEM_RESET, STRING_DATA, ANALOG_ME↩ SSAGE, DIGITAL_MESSAGE, etc).

**Parameters**

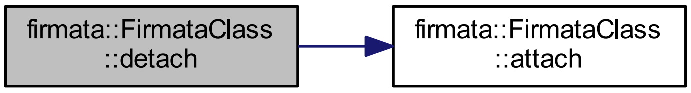
| | |
|---|---|
| *command* | The ID of the command to detatch the callback function from. |

Definition at line 452 of file Firmata.cpp.

```
{
  switch (command) {
    case SYSTEM_RESET:
      attach(command, (systemCallbackFunction)NULL);
      break;
    case STRING_DATA:
      attach(command, (stringCallbackFunction)NULL);
      break;
    case START_SYSEX:
      attach(command, (sysexCallbackFunction)NULL);
      break;
    default:
      attach(command, (callbackFunction)NULL);
      break;
  }
}
```

References attach(), firmata::START_SYSEX, firmata::STRING_DATA, and firmata::SYSTEM_RESET.

Here is the call graph for this function:


firmata::FirmataClass ::detach
firmata::FirmataClass ::attach

### 26.5.4.11 disableBlinkVersion()

```
void FirmataClass::disableBlinkVersion ( )
```

Provides a means to disable the version blink sequence on the onboard LED, trimming startup time by a couple of seconds. Call this before Firmata.begin(). It only applies when using the default Serial transport.

Definition at line 177 of file Firmata.cpp.

```
{
  blinkVersionDisabled = true;
}
```

#### 26.5.4.12 endSysex()

```
void FirmataClass::endSysex (
             void )
```

A helper method to write the end of a Sysex message transmission.

Definition at line 67 of file Firmata.cpp.

```
{
  FirmataStream->write(END_SYSEX);
}
```

References firmata::END_SYSEX.

#### 26.5.4.13 getPinMode()

```
byte FirmataClass::getPinMode (
             byte pin )
```

**Parameters**

| | |
|---|---|
| *pin* | The pin to get the configuration of. |

**Returns**

The configuration of the specified pin.

Definition at line 474 of file Firmata.cpp.

```
{
  return pinConfig[pin];
}
```

#### 26.5.4.14 getPinState()

```
int FirmataClass::getPinState (
             byte pin )
```

**Parameters**

| | |
|---|---|
| *pin* | The pin to get the state of. |

**Returns**

The state of the specified pin.

Definition at line 498 of file Firmata.cpp.

```
{
  return pinState[pin];
}
```

#### 26.5.4.15 isParsingMessage()

```
boolean FirmataClass::isParsingMessage (
             void  )
```

**Returns**

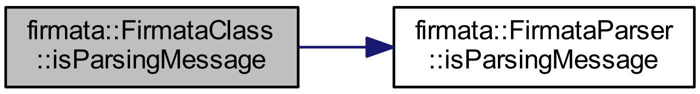
Returns true if the parser is actively parsing data.

Definition at line 272 of file Firmata.cpp.

```
{
  return parser.isParsingMessage();
}
```

References firmata::FirmataParser::isParsingMessage().

Here is the call graph for this function:



#### 26.5.4.16 parse()

```
void FirmataClass::parse (
             unsigned char value )
```

Parse data from the input stream.

**Parameters**

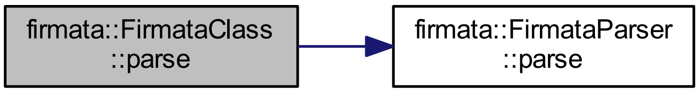
| | |
|---|---|
| *inputData* | A single byte to be added to the parser. |

Definition at line 264 of file Firmata.cpp.

```
{
    parser.parse(inputData);
}
```

References firmata::FirmataParser::parse().

Here is the call graph for this function:



#### 26.5.4.17 printFirmwareVersion()

```
void FirmataClass::printFirmwareVersion (
            void  )
```

Sends the firmware name and version to the Firmata host application. The major and minor version numbers are the first 2 bytes in the message. The following bytes are the characters of the firmware name.

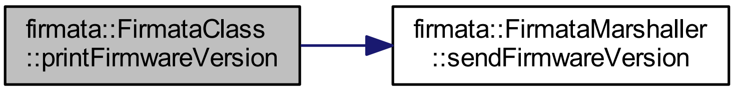
Definition at line 187 of file Firmata.cpp.

```
{
  if (firmwareVersionCount) { // make sure that the name has been set before reporting
    marshaller.sendFirmwareVersion(static_cast<uint8_t>(firmwareVersionVector[0]),
   static_cast<uint8_t>(firmwareVersionVector[1]), (firmwareVersionCount - 2), reinterpret_cast<uint8_t
   *>(&firmwareVersionVector[2]));
  }
}
```

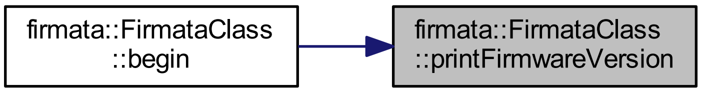
References firmata::FirmataMarshaller::sendFirmwareVersion().

Referenced by begin().

Here is the call graph for this function:


firmata::FirmataClass ::printFirmwareVersion
firmata::FirmataMarshaller ::sendFirmwareVersion

Here is the caller graph for this function:


firmata::FirmataClass ::begin
firmata::FirmataClass ::printFirmwareVersion

#### 26.5.4.18 printVersion()

```
void FirmataClass::printVersion (
             void   )
```

Send the Firmata protocol version to the Firmata host application.

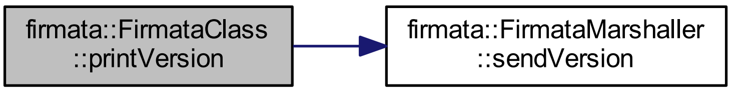
Definition at line 147 of file Firmata.cpp.

```
{
  marshaller.sendVersion(FIRMATA_PROTOCOL_MAJOR_VERSION, FIRMATA_PROTOCOL_MINOR_VERSION);
}
```

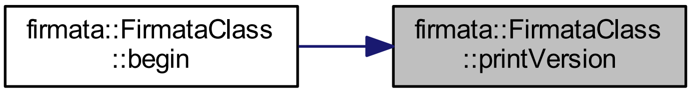
References FIRMATA_PROTOCOL_MAJOR_VERSION, FIRMATA_PROTOCOL_MINOR_VERSION, and firmata::FirmataMarshaller::sendVersion().

Referenced by begin().

Here is the call graph for this function:


firmata::FirmataClass ::printVersion
firmata::FirmataMarshaller ::sendVersion

Here is the caller graph for this function:


firmata::FirmataClass ::begin
firmata::FirmataClass ::printVersion

#### 26.5.4.19 processInput()

```
void FirmataClass::processInput (
             void )
```

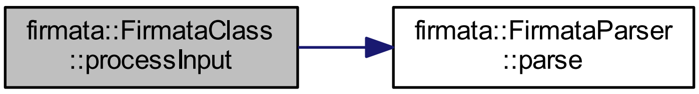
Read a single int from the input stream. If the value is not = -1, pass it on to parse(byte)

Definition at line 252 of file Firmata.cpp.

```
{
  int inputData = FirmataStream->read(); // this is 'int' to handle -1 when no data
  if (inputData != -1) {
    parser.parse(inputData);
  }
}
```

References firmata::FirmataParser::parse().

Here is the call graph for this function:


firmata::FirmataClass ::processInput
firmata::FirmataParser ::parse

#### 26.5.4.20 sendAnalog()

```
void FirmataClass::sendAnalog (
             byte pin,
             int value )
```

Send an analog message to the Firmata host application. The range of pins is limited to [0..15] when using the ANALOG_MESSAGE. The maximum value of the ANALOG_MESSAGE is limited to 14 bits (16384). To increase the pin range or value, see the documentation for the EXTENDED_ANALOG message.

**Parameters**

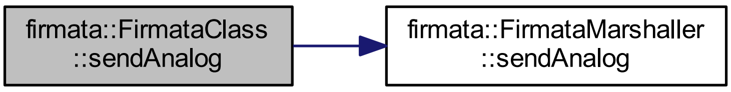
| | |
|---|---|
| *pin* | The analog pin to send the value of (limited to pins 0 - 15). |
| *value* | The value of the analog pin (0 - 1024 for 10-bit analog, 0 - 4096 for 12-bit, etc). The maximum value is 14-bits (16384). |

Definition at line 289 of file Firmata.cpp.

```
{
  marshaller.sendAnalog(pin, value);
}
```

References firmata::FirmataMarshaller::sendAnalog().

Here is the call graph for this function:


firmata::FirmataClass ::sendAnalog
firmata::FirmataMarshaller ::sendAnalog

#### 26.5.4.21 sendDigital()

```
void FirmataClass::sendDigital (
              byte pin,
              int value )
```

Definition at line 300 of file Firmata.cpp.

```
{
  (void)pin;
  (void)value;
  /* TODO add single pin digital messages to the protocol, this needs to
   * track the last digital data sent so that it can be sure to change just
   * one bit in the packet.  This is complicated by the fact that the
   * numbering of the pins will probably differ on Arduino, Wiring, and
   * other boards.
   */

  // TODO: the digital message should not be sent on the serial port every
  // time sendDigital() is called.  Instead, it should add it to an int
  // which will be sent on a schedule.  If a pin changes more than once
  // before the digital message is sent on the serial port, it should send a
  // digital message for each change.

  //    if(value == 0)
  //        sendDigitalPortPair();
}
```

#### 26.5.4.22 sendDigitalPort()

```
void FirmataClass::sendDigitalPort (
              byte portNumber,
              int portData )
```

Send an 8-bit port in a single digital message (protocol v2 and later). Send 14-bits in a single digital message (protocol v1).

**Parameters**

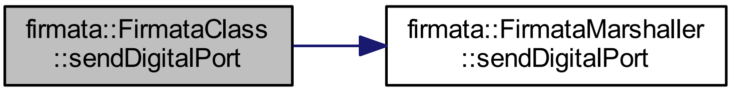
| | |
|---|---|
| *portNumber* | The port number to send. Note that this is not the same as a "port" on the physical microcontroller. Ports are defined in order per every 8 pins in ascending order of the Arduino digital pin numbering scheme. Port 0 = pins D0 - D7, port 1 = pins D8 - D15, etc. |
| *portData* | The value of the port. The value of each pin in the port is represented by a bit. |

Definition at line 330 of file Firmata.cpp.

```
{
  marshaller.sendDigitalPort(portNumber, portData);
}
```

References firmata::FirmataMarshaller::sendDigitalPort().

Here is the call graph for this function:


firmata::FirmataClass ::sendDigitalPort
firmata::FirmataMarshaller ::sendDigitalPort

#### 26.5.4.23 sendString() `[1/2]`

```
void FirmataClass::sendString (
            byte command,
            const char * string )
```

Send a string to the Firmata host application.

**Parameters**

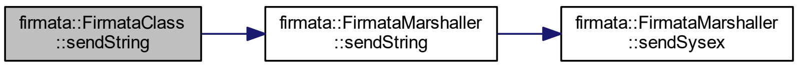
| | |
|---|---|
| *command* | Must be STRING_DATA |
| *string* | A pointer to the char string |

Definition at line 352 of file Firmata.cpp.

```
{
  if (command == STRING_DATA) {
    marshaller.sendString(string);
  }
}
```

References firmata::FirmataMarshaller::sendString(), and firmata::STRING_DATA.

Here is the call graph for this function:


firmata::FirmataClass ::sendString
firmata::FirmataMarshaller ::sendString
firmata::FirmataMarshaller ::sendSysex

### 26.5.4.24 sendString() `[2/2]`

```
void FirmataClass::sendString (
             const char * string )
```

Send a string to the Firmata host application.

**Parameters**

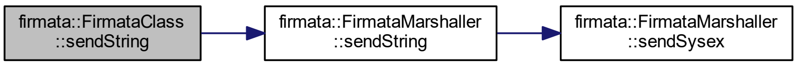
| | |
|---|---|
| *string* | A pointer to the char string |

Definition at line 363 of file Firmata.cpp.

```
{
  marshaller.sendString(string);
}
```

References firmata::FirmataMarshaller::sendString().

Here is the call graph for this function:


firmata::FirmataClass ::sendString
firmata::FirmataMarshaller ::sendString
firmata::FirmataMarshaller ::sendSysex

### 26.5.4.25 sendSysex()

```
void FirmataClass::sendSysex (
             byte command,
             byte bytec,
             byte * bytev )
```

Send a sysex message where all values after the command byte are packet as 2 7-bit bytes (this is not always the case so this function is not always used to send sysex messages).

**Parameters**

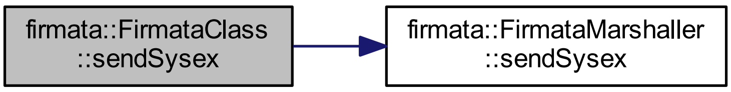
| | |
|---|---|
| *command* | The sysex command byte. |
| *bytec* | The number of data bytes in the message (excludes start, command and end bytes). |
| *bytev* | A pointer to the array of data bytes to send in the message. |

Definition at line 342 of file Firmata.cpp.

```
{
  marshaller.sendSysex(command, bytec, bytev);
}
```

References firmata::FirmataMarshaller::sendSysex().

Here is the call graph for this function:


firmata::FirmataClass ::sendSysex
firmata::FirmataMarshaller ::sendSysex

#### 26.5.4.26 sendValueAsTwo7bitBytes()

```
void FirmataClass::sendValueAsTwo7bitBytes (
             int value )
```

Split a 16-bit byte into two 7-bit values and write each value.

**Parameters**

| | |
|---|---|
| *value* | The 16-bit value to be split and written separately. |

Definition at line 51 of file Firmata.cpp.

```
{
  marshaller.encodeByteStream(sizeof(value), reinterpret_cast<uint8_t *>(&value), sizeof(value));
}
```

#### 26.5.4.27 setFirmwareNameAndVersion()

```
void FirmataClass::setFirmwareNameAndVersion (
             const char * name,
             byte major,
             byte minor )
```

Sets the name and version of the firmware. This is not the same version as the Firmata protocol (although at times the firmware version and protocol version may be the same number).

**Parameters**

| | |
|---|---|
| *name* | A pointer to the name char array |
| *major* | The major version number |
| *minor* | The minor version number |

Definition at line 201 of file Firmata.cpp.

```
{
  const char *firmwareName;
  const char *extension;

  // parse out ".cpp" and "applet/" that comes from using __FILE__
  extension = strstr(name, ".cpp");
  firmwareName = strrchr(name, '/');

  if (!firmwareName) {
    // windows
    firmwareName = strrchr(name, '\\');
  }
  if (!firmwareName) {
    // user passed firmware name
    firmwareName = name;
  } else {
    firmwareName ++;
  }

  if (!extension) {
    firmwareVersionCount = strlen(firmwareName) + 2;
  } else {
    firmwareVersionCount = extension - firmwareName + 2;
  }

  // in case anyone calls setFirmwareNameAndVersion more than once
  free(firmwareVersionVector);

  firmwareVersionVector = (byte *) malloc(firmwareVersionCount + 1);
  firmwareVersionVector[firmwareVersionCount] = 0;
  firmwareVersionVector[0] = major;
  firmwareVersionVector[1] = minor;
  strncpy((char *)firmwareVersionVector + 2, firmwareName, firmwareVersionCount - 2);
}
```

### 26.5.4.28 setPinMode()

```
void FirmataClass::setPinMode (
            byte pin,
            byte config )
```

Set the pin mode/configuration. The pin configuration (or mode) in Firmata represents the current function of the pin. Examples are digital input or output, analog input, pwm, i2c, serial (uart), etc.

**Parameters**

| | |
|---|---|
| *pin* | The pin to configure. |
| *config* | The configuration value for the specified pin. |

Definition at line 486 of file Firmata.cpp.

```
{
  if (pinConfig[pin] == PIN_MODE_IGNORE)
    return;

  pinConfig[pin] = config;
}
```

References firmata::PIN_MODE_IGNORE.

### 26.5.4.29 setPinState()

```
void FirmataClass::setPinState (
            byte pin,
            int state )
```

Set the pin state. The pin state of an output pin is the pin value. The state of an input pin is 0, unless the pin has it's internal pull up resistor enabled, then the value is 1.

**Parameters**

| | |
|---|---|
| *pin* | The pin to set the state of |
| *state* | Set the state of the specified pin |

Definition at line 509 of file Firmata.cpp.

```
{
  pinState[pin] = state;
}
```

#### 26.5.4.30 startSysex()

```
void FirmataClass::startSysex (
            void  )
```

A helper method to write the beginning of a Sysex message transmission.

Definition at line 59 of file Firmata.cpp.

```
{
  FirmataStream->write(START_SYSEX);
}
```

References firmata::START_SYSEX.

#### 26.5.4.31 write()

```
void FirmataClass::write (
            byte c )
```

A wrapper for Stream::available(). Write a single byte to the output stream.

**Parameters**

| | |
|---|---|
| *c* | The byte to be written. |

Definition at line 373 of file Firmata.cpp.

```
{
  FirmataStream->write(c);
}
```

### 26.5.5 Friends And Related Function Documentation

#### 26.5.5.1 FirmataMarshaller::encodeByteStream

```
void FirmataMarshaller::encodeByteStream (
            size_t bytec,
            uint8_t * bytev,
            size_t max_bytes ) const  [friend]
```

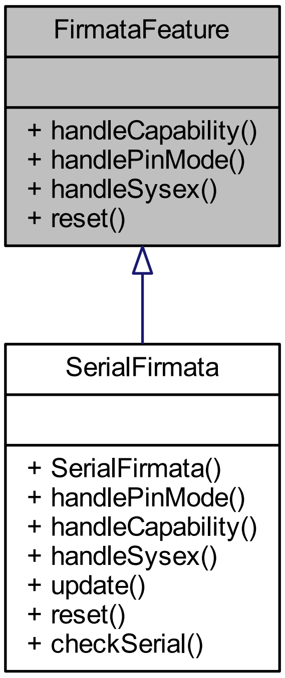
The documentation for this class was generated from the following files:

- Firmata.h
- Firmata.cpp

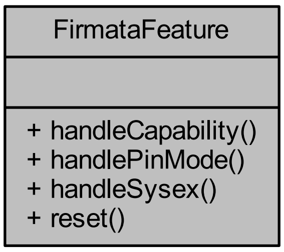
## 26.6 FirmataFeature Class Reference

```
#include <FirmataFeature.h>
```

Inheritance diagram for FirmataFeature:



Collaboration diagram for FirmataFeature:



## Public Member Functions

- virtual void handleCapability (byte pin)=0
- virtual boolean handlePinMode (byte pin, int mode)=0
- virtual boolean handleSysex (byte command, byte argc, byte ∗argv)=0
- virtual void reset ()=0

## 26.6.1 Detailed Description

Definition at line 29 of file FirmataFeature.h.

## 26.6.2 Member Function Documentation

### 26.6.2.1 handleCapability()

```
virtual void FirmataFeature::handleCapability (
            byte pin )  [pure virtual]
```

Implemented in SerialFirmata.

### 26.6.2.2 handlePinMode()

```
virtual boolean FirmataFeature::handlePinMode (
            byte pin,
            int mode )  [pure virtual]
```

Implemented in SerialFirmata.

#### 26.6.2.3 handleSysex()

```
virtual boolean FirmataFeature::handleSysex (
            byte command,
            byte argc,
            byte * argv )  [pure virtual]
```

Implemented in SerialFirmata.

#### 26.6.2.4 reset()

```
virtual void FirmataFeature::reset ( )  [pure virtual]
```

Implemented in SerialFirmata.

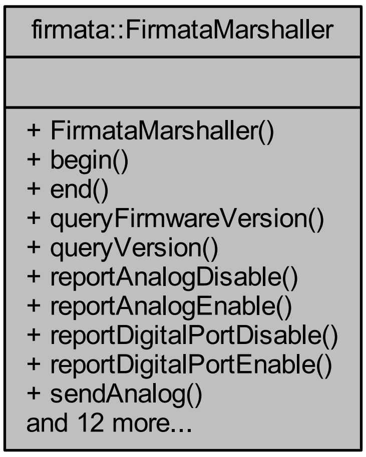
The documentation for this class was generated from the following file:

- FirmataFeature.h

## 26.7 firmata::FirmataMarshaller Class Reference

```
#include <FirmataMarshaller.h>
```

Collaboration diagram for firmata::FirmataMarshaller:



## Public Member Functions

- FirmataMarshaller ()
- void begin (Stream &s)
- void end ()
- void queryFirmwareVersion (void) const
- void queryVersion (void) const
- void reportAnalogDisable (uint8_t pin) const
- void reportAnalogEnable (uint8_t pin) const
- void reportDigitalPortDisable (uint8_t portNumber) const
- void reportDigitalPortEnable (uint8_t portNumber) const
- void sendAnalog (uint8_t pin, uint16_t value) const
- void sendAnalogMappingQuery (void) const
- void sendCapabilityQuery (void) const
- void sendDigital (uint8_t pin, uint8_t value) const
- void sendDigitalPort (uint8_t portNumber, uint16_t portData) const
- void sendFirmwareVersion (uint8_t major, uint8_t minor, size_t bytec, uint8_t ∗bytev) const
- void sendVersion (uint8_t major, uint8_t minor) const
- void sendPinMode (uint8_t pin, uint8_t config) const
- void sendPinStateQuery (uint8_t pin) const
- void sendString (const char ∗string) const
- void sendSysex (uint8_t command, size_t bytec, uint8_t ∗bytev) const
- void setSamplingInterval (uint16_t interval_ms) const
- void systemReset (void) const

## Friends

- class FirmataClass

## 26.7.1 Detailed Description

Definition at line 29 of file FirmataMarshaller.h.

## 26.7.2 Constructor & Destructor Documentation

### 26.7.2.1 FirmataMarshaller()

```
FirmataMarshaller::FirmataMarshaller ( )
```

The FirmataMarshaller class.

Definition at line 129 of file FirmataMarshaller.cpp.

```
130 :
131   FirmataStream((Stream *)NULL)
132 {
133 }
```

### 26.7.3 Member Function Documentation

#### 26.7.3.1 begin()

```
void FirmataMarshaller::begin (
             Stream & s )
```

Reassign the Firmata stream transport.

**Parameters**

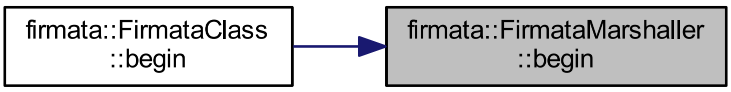
| | |
|---|---|
| *s* | A reference to the Stream transport object. This can be any type of transport that implements the Stream interface. Some examples include Ethernet, WiFi and other UARTs on the board (Serial1, Serial2, etc). |

Definition at line 145 of file FirmataMarshaller.cpp.

```
{
  FirmataStream = &s;
}
```

Referenced by firmata::FirmataClass::begin().

Here is the caller graph for this function:


firmata::FirmataClass ::begin
firmata::FirmataMarshaller ::begin

### 26.7.3.2 end()

```
void FirmataMarshaller::end (
             void  )
```

Closes the FirmataMarshaller stream by setting its stream reference to `(Stream *)NULL`

Definition at line 153 of file FirmataMarshaller.cpp.

```
{
  FirmataStream = (Stream *)NULL;
}
```

### 26.7.3.3 queryFirmwareVersion()

```
void FirmataMarshaller::queryFirmwareVersion (
             void  ) const
```

Query the target's firmware name and version

Definition at line 165 of file FirmataMarshaller.cpp.

```
{
  if ( (Stream *)NULL == FirmataStream ) { return; }
  FirmataStream->write(START_SYSEX);
  FirmataStream->write(REPORT_FIRMWARE);
  FirmataStream->write(END_SYSEX);
}
```

References firmata::END_SYSEX, firmata::REPORT_FIRMWARE, and firmata::START_SYSEX.

#### 26.7.3.4 queryVersion()

```
void FirmataMarshaller::queryVersion (
            void ) const
```

Query the target's Firmata protocol version

Definition at line 177 of file FirmataMarshaller.cpp.

```
{
  if ( (Stream *)NULL == FirmataStream ) { return; }
  FirmataStream->write(REPORT_VERSION);
}
```

References firmata::REPORT_VERSION.

#### 26.7.3.5 reportAnalogDisable()

```
void FirmataMarshaller::reportAnalogDisable (
            uint8_t pin ) const
```

Halt the stream of analog readings from the Firmata host application. The range of pins is limited to [0..15] when using the REPORT_ANALOG. The maximum result of the REPORT_ANALOG is limited to 14 bits (16384). To increase the pin range or value, see the documentation for the EXTENDED_ANALOG message.

**Parameters**

| | |
|---|---|
| *pin* | The analog pin for which to request the value (limited to pins 0 - 15). |

Definition at line 191 of file FirmataMarshaller.cpp.

```
{
  reportAnalog(pin, false);
}
```

#### 26.7.3.6 reportAnalogEnable()

```
void FirmataMarshaller::reportAnalogEnable (
            uint8_t pin ) const
```

Request a stream of analog readings from the Firmata host application. The range of pins is limited to [0..15] when using the REPORT_ANALOG. The maximum result of the REPORT_ANALOG is limited to 14 bits (16384). To increase the pin range or value, see the documentation for the EXTENDED_ANALOG message.

**Parameters**

| | |
|---|---|
| *pin* | The analog pin for which to request the value (limited to pins 0 - 15). |

Definition at line 204 of file FirmataMarshaller.cpp.

```
{
  reportAnalog(pin, true);
}
```

#### 26.7.3.7 reportDigitalPortDisable()

```
void FirmataMarshaller::reportDigitalPortDisable (
            uint8_t portNumber ) const
```

Halt an 8-bit port stream from the Firmata host application (protocol v2 and later). Send 14-bits in a single digital message (protocol v1).

**Parameters**

| | |
|---|---|
| *portNumber* | The port number for which to request the value. Note that this is not the same as a "port" on the physical microcontroller. Ports are defined in order per every 8 pins in ascending order of the Arduino digital pin numbering scheme. Port 0 = pins D0 - D7, port 1 = pins D8 - D15, etc. |

Definition at line 217 of file FirmataMarshaller.cpp.

```
{
  reportDigitalPort(portNumber, false);
}
```

#### 26.7.3.8 reportDigitalPortEnable()

```
void FirmataMarshaller::reportDigitalPortEnable (
            uint8_t portNumber ) const
```

Request an 8-bit port stream from the Firmata host application (protocol v2 and later). Send 14-bits in a single digital message (protocol v1).

**Parameters**

| | |
|---|---|
| *portNumber* | The port number for which to request the value. Note that this is not the same as a "port" on the physical microcontroller. Ports are defined in order per every 8 pins in ascending order of the Arduino digital pin numbering scheme. Port 0 = pins D0 - D7, port 1 = pins D8 - D15, etc. |

Definition at line 230 of file FirmataMarshaller.cpp.

```
{
  reportDigitalPort(portNumber, true);
}
```

#### 26.7.3.9 sendAnalog()

```
void FirmataMarshaller::sendAnalog (
            uint8_t pin,
            uint16_t value ) const
```

Send an analog message to the Firmata host application. The range of pins is limited to [0..15] when using the ANALOG_MESSAGE. The maximum value of the ANALOG_MESSAGE is limited to 14 bits (16384). To increase the pin range or value, see the documentation for the EXTENDED_ANALOG message.

**Parameters**

| | |
|---|---|
| *pin* | The analog pin to which the value is sent. |
| *value* | The value of the analog pin (0 - 1024 for 10-bit analog, 0 - 4096 for 12-bit, etc). |

**Note**

The maximum value is 14-bits (16384).

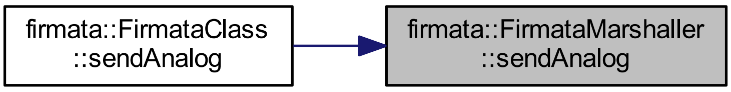
Definition at line 245 of file FirmataMarshaller.cpp.

```
{
  if ( (Stream *)NULL == FirmataStream ) { return; }
  if ( (0xF >= pin) && (0x3FFF >= value) ) {
    FirmataStream->write(ANALOG_MESSAGE|pin);
    encodeByteStream(sizeof(value), reinterpret_cast<uint8_t *>(&value), sizeof(value));
  } else {
    sendExtendedAnalog(pin, sizeof(value), reinterpret_cast<uint8_t *>(&value));
  }
}
```

References firmata::ANALOG_MESSAGE.

Referenced by firmata::FirmataClass::sendAnalog().

Here is the caller graph for this function:


firmata::FirmataClass ::sendAnalog
firmata::FirmataMarshaller ::sendAnalog

### 26.7.3.10 sendAnalogMappingQuery()

```
void FirmataMarshaller::sendAnalogMappingQuery (
            void  ) const
```

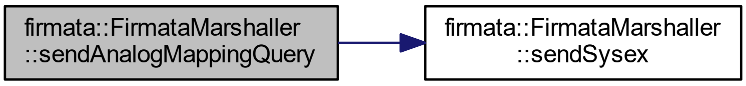
Send an analog mapping query to the Firmata host application. The resulting sysex message will have an A←NALOG_MAPPING_RESPONSE command byte, followed by a list of pins [0-n]; where each pin will specify its corresponding analog pin number or 0x7F (127) if not applicable.

Definition at line 262 of file FirmataMarshaller.cpp.

```
{
  sendSysex(ANALOG_MAPPING_QUERY, 0, NULL);
}
```

References firmata::ANALOG_MAPPING_QUERY, and sendSysex().

Here is the call graph for this function:


firmata::FirmataMarshaller ::sendAnalogMappingQuery
firmata::FirmataMarshaller ::sendSysex

### 26.7.3.11 sendCapabilityQuery()

```
void FirmataMarshaller::sendCapabilityQuery (
            void  ) const
```

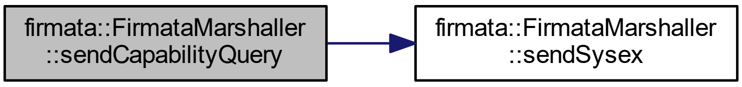
Send a capability query to the Firmata host application. The resulting sysex message will have a CAPABILITY_↩ RESPONSE command byte, followed by a list of byte tuples (mode and mode resolution) for each pin; where each pin list is terminated by 0x7F (127).

Definition at line 273 of file FirmataMarshaller.cpp.

```
{
  sendSysex(CAPABILITY_QUERY, 0, NULL);
}
```

References firmata::CAPABILITY_QUERY, and sendSysex().

Here is the call graph for this function:


firmata::FirmataMarshaller ::sendCapabilityQuery
firmata::FirmataMarshaller ::sendSysex

### 26.7.3.12 sendDigital()

```
void FirmataMarshaller::sendDigital (
            uint8_t pin,
            uint8_t value ) const
```

Send a single digital pin value to the Firmata host application.

**Parameters**

| | |
|---|---|
| *pin* | The digital pin to send the value of. |
| *value* | The value of the pin. |

Definition at line 284 of file FirmataMarshaller.cpp.

```
{
  if ( (Stream *)NULL == FirmataStream ) { return; }
  FirmataStream->write(SET_DIGITAL_PIN_VALUE);
  FirmataStream->write(pin & 0x7F);
  FirmataStream->write(value != 0);
}
```

References firmata::SET_DIGITAL_PIN_VALUE.

### 26.7.3.13 sendDigitalPort()

```
void FirmataMarshaller::sendDigitalPort (
            uint8_t portNumber,
            uint16_t portData ) const
```

Send an 8-bit port in a single digital message (protocol v2 and later). Send 14-bits in a single digital message (protocol v1).

**Parameters**

| | |
|---|---|
| *portNumber* | The port number to send. Note that this is not the same as a "port" on the physical microcontroller. Ports are defined in order per every 8 pins in ascending order of the Arduino digital pin numbering scheme. Port 0 = pins D0 - D7, port 1 = pins D8 - D15, etc. |
| *portData* | The value of the port. The value of each pin in the port is represented by a bit. |

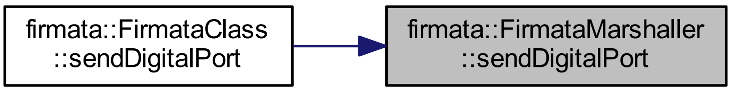
Definition at line 302 of file FirmataMarshaller.cpp.

```
{
  if ( (Stream *)NULL == FirmataStream ) { return; }
  FirmataStream->write(DIGITAL_MESSAGE | (portNumber & 0xF));
  // Tx bits  0-6 (protocol v1 and higher)
  // Tx bits 7-13 (bit 7 only for protocol v2 and higher)
  encodeByteStream(sizeof(portData), reinterpret_cast<uint8_t *>(&portData), sizeof(portData));
}
```

References firmata::DIGITAL_MESSAGE.

Referenced by firmata::FirmataClass::sendDigitalPort().

Here is the caller graph for this function:


firmata::FirmataClass ::sendDigitalPort
firmata::FirmataMarshaller ::sendDigitalPort

### 26.7.3.14 sendFirmwareVersion()

```
void FirmataMarshaller::sendFirmwareVersion (
            uint8_t major,
            uint8_t minor,
            size_t bytec,
            uint8_t * bytev ) const
```

Sends the firmware name and version to the Firmata host application.

**Parameters**

| | |
|---|---|
| *major* | The major verison number |
| *minor* | The minor version number |
| *bytec* | The length of the firmware name |
| *bytev* | The firmware name array |

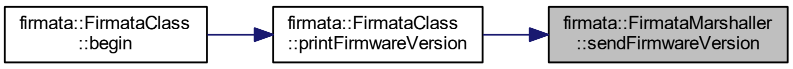
Definition at line 319 of file FirmataMarshaller.cpp.

```
{
  if ( (Stream *)NULL == FirmataStream ) { return; }
  size_t i;
  FirmataStream->write(START_SYSEX);
  FirmataStream->write(REPORT_FIRMWARE);
  FirmataStream->write(major);
  FirmataStream->write(minor);
  for (i = 0; i < bytec; ++i) {
    encodeByteStream(sizeof(bytev[i]), reinterpret_cast<uint8_t *>(&bytev[i]));
  }
  FirmataStream->write(END_SYSEX);
}
```

References firmata::END_SYSEX, firmata::REPORT_FIRMWARE, and firmata::START_SYSEX.

Referenced by firmata::FirmataClass::printFirmwareVersion().

Here is the caller graph for this function:


firmata::FirmataClass ::begin
firmata::FirmataClass ::printFirmwareVersion
firmata::FirmataMarshaller ::sendFirmwareVersion

### 26.7.3.15 sendPinMode()

```
void FirmataMarshaller::sendPinMode (
            uint8_t pin,
            uint8_t config ) const
```

Send the pin mode/configuration. The pin configuration (or mode) in Firmata represents the current function of the pin. Examples are digital input or output, analog input, pwm, i2c, serial (uart), etc.

**Parameters**

| | |
|---|---|
| *pin* | The pin to configure. |
| *config* | The configuration value for the specified pin. |

Definition at line 355 of file FirmataMarshaller.cpp.

```
{
  if ( (Stream *)NULL == FirmataStream ) { return; }
  FirmataStream->write(SET_PIN_MODE);
  FirmataStream->write(pin);
  FirmataStream->write(config);
}
```

References firmata::SET_PIN_MODE.

### 26.7.3.16 sendPinStateQuery()

```
void FirmataMarshaller::sendPinStateQuery (
            uint8_t pin ) const
```

Send a pin state query to the Firmata host application. The resulting sysex message will have a PIN_STATE_RE↩SPONSE command byte, followed by the pin number, the pin mode and a stream of bits to indicate any *data* written to the pin (pin state).

**Parameters**

| | |
|---|---|
| *pin* | The pin to query |

**Note**

The pin state is any data written to the pin (i.e. pin state != pin value)

Definition at line 371 of file FirmataMarshaller.cpp.

```
{
  if ( (Stream *)NULL == FirmataStream ) { return; }
  FirmataStream->write(START_SYSEX);
  FirmataStream->write(PIN_STATE_QUERY);
  FirmataStream->write(pin);
  FirmataStream->write(END_SYSEX);
}
```

References firmata::END_SYSEX, firmata::PIN_STATE_QUERY, and firmata::START_SYSEX.

### 26.7.3.17 sendString()

```
void FirmataMarshaller::sendString (
            const char * string ) const
```

Send a string to the Firmata host application.

**Parameters**

| | |
|---|---|
| *string* | A pointer to the char string |

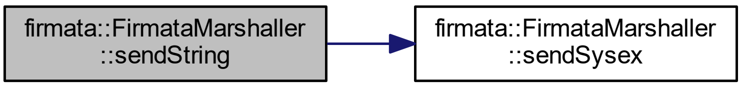
Definition at line 405 of file FirmataMarshaller.cpp.

```
{
  sendSysex(STRING_DATA, strlen(string), reinterpret_cast<uint8_t *>(const_cast<char *>(string)));
}
```

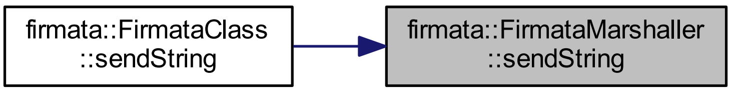
References sendSysex(), and firmata::STRING_DATA.

Referenced by firmata::FirmataClass::sendString().

Here is the call graph for this function:


firmata::FirmataMarshaller ::sendString
firmata::FirmataMarshaller ::sendSysex

Here is the caller graph for this function:


firmata::FirmataClass ::sendString
firmata::FirmataMarshaller ::sendString

### 26.7.3.18 sendSysex()

```
void FirmataMarshaller::sendSysex (
            uint8_t command,
            size_t bytec,
            uint8_t * bytev ) const
```

Send a sysex message where all values after the command byte are packet as 2 7-bit bytes (this is not always the case so this function is not always used to send sysex messages).

**Parameters**

| | |
|---|---|
| *command* | The sysex command byte. |
| *bytec* | The number of data bytes in the message (excludes start, command and end bytes). |
| *bytev* | A pointer to the array of data bytes to send in the message. |

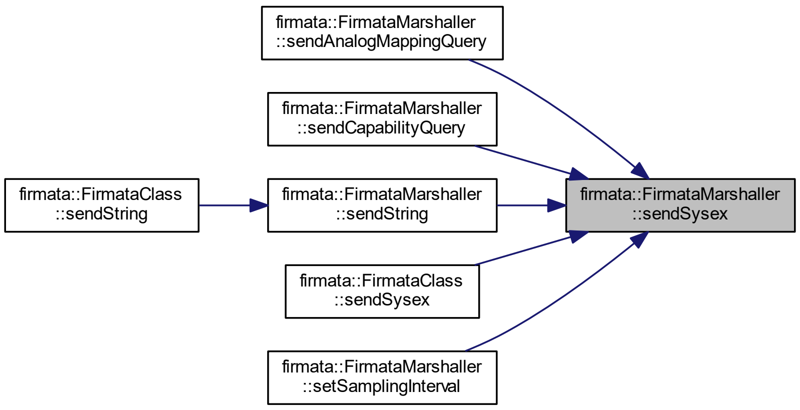
Definition at line 388 of file FirmataMarshaller.cpp.

```
{
  if ( (Stream *)NULL == FirmataStream ) { return; }
  size_t i;
  FirmataStream->write(START_SYSEX);
  FirmataStream->write(command);
  for (i = 0; i < bytec; ++i) {
    encodeByteStream(sizeof(bytev[i]), reinterpret_cast<uint8_t *>(&bytev[i]));
  }
  FirmataStream->write(END_SYSEX);
}
```

References firmata::END_SYSEX, and firmata::START_SYSEX.

Referenced by sendAnalogMappingQuery(), sendCapabilityQuery(), sendString(), firmata::FirmataClass::send↩ Sysex(), and setSamplingInterval().

Here is the caller graph for this function:


firmata::FirmataMarshaller ::sendAnalogMappingQuery
firmata::FirmataMarshaller ::sendCapabilityQuery
firmata::FirmataClass ::sendString
firmata::FirmataMarshaller ::sendString
firmata::FirmataMarshaller ::sendSysex
firmata::FirmataClass ::sendSysex
firmata::FirmataMarshaller ::setSamplingInterval

### 26.7.3.19 sendVersion()

```
void FirmataMarshaller::sendVersion (
            uint8_t major,
            uint8_t minor ) const
```

Send the Firmata protocol version to the Firmata host application.

**Parameters**

| | |
|---|---|
| *major* | The major verison number |
| *minor* | The minor version number |

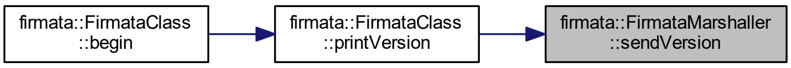
Definition at line 339 of file FirmataMarshaller.cpp.

```
{
  if ( (Stream *)NULL == FirmataStream ) { return; }
  FirmataStream->write(REPORT_VERSION);
  FirmataStream->write(major);
  FirmataStream->write(minor);
}
```

References firmata::REPORT_VERSION.

Referenced by firmata::FirmataClass::printVersion().

Here is the caller graph for this function:


firmata::FirmataClass ::begin
firmata::FirmataClass ::printVersion
firmata::FirmataMarshaller ::sendVersion

### 26.7.3.20 setSamplingInterval()

```
void FirmataMarshaller::setSamplingInterval (
            uint16_t interval_ms ) const
```

The sampling interval sets how often analog data and i2c data is reported to the client.

**Parameters**

| | |
|---|---|
| *interval_ms* | The interval (in milliseconds) at which to sample |

**Note**

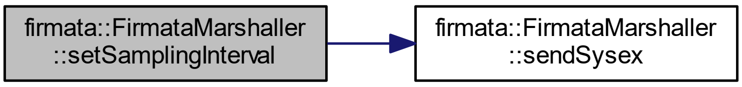
The default sampling interval is 19ms

Definition at line 416 of file FirmataMarshaller.cpp.

```
{
  sendSysex(SAMPLING_INTERVAL, sizeof(interval_ms), reinterpret_cast<uint8_t *>(&interval_ms));
}
```

References firmata::SAMPLING_INTERVAL, and sendSysex().

Here is the call graph for this function:


firmata::FirmataMarshaller ::setSamplingInterval
firmata::FirmataMarshaller ::sendSysex

#### 26.7.3.21 systemReset()

```
void FirmataMarshaller::systemReset (
            void  ) const
```

Perform a software reset on the target. For example, StandardFirmata.ino will initialize everything to a known state and reset the parsing buffer.

Definition at line 426 of file FirmataMarshaller.cpp.

```
{
  if ( (Stream *)NULL == FirmataStream ) { return; }
  FirmataStream->write(SYSTEM_RESET);
}
```

References firmata::SYSTEM_RESET.

### 26.7.4 Friends And Related Function Documentation

#### 26.7.4.1 FirmataClass

```
friend class FirmataClass  [friend]
```

Definition at line 31 of file FirmataMarshaller.h.

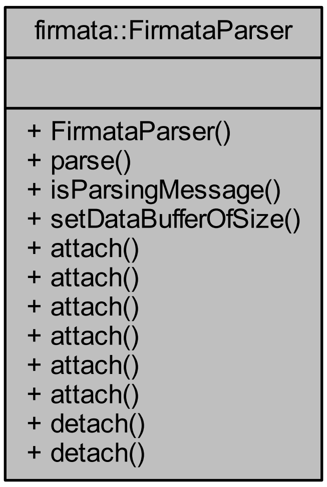
The documentation for this class was generated from the following files:

- FirmataMarshaller.h
- FirmataMarshaller.cpp

## 26.8 firmata::FirmataParser Class Reference

```
#include <FirmataParser.h>
```

Collaboration diagram for firmata::FirmataParser:



## Public Types

- typedef void(∗ callbackFunction) (void ∗context, uint8_t command, uint16_t value)
- typedef void(∗ dataBufferOverflowCallbackFunction) (void ∗context)
- typedef void(∗ stringCallbackFunction) (void ∗context, const char ∗c_str)
- typedef void(∗ sysexCallbackFunction) (void ∗context, uint8_t command, size_t argc, uint8_t ∗argv)
- typedef void(∗ systemCallbackFunction) (void ∗context)
- typedef void(∗ versionCallbackFunction) (void ∗context, size_t sv_major, size_t sv_minor, const char ∗firmware)

## Public Member Functions

- FirmataParser (uint8_t ∗dataBuffer=(uint8_t ∗) NULL, size_t dataBufferSize=0)
- void parse (uint8_t value)
- bool isParsingMessage (void) const
- int setDataBufferOfSize (uint8_t ∗dataBuffer, size_t dataBufferSize)
- void attach (uint8_t command, callbackFunction newFunction, void ∗context=NULL)
- void attach (dataBufferOverflowCallbackFunction newFunction, void ∗context=NULL)
- void attach (uint8_t command, stringCallbackFunction newFunction, void ∗context=NULL)
- void attach (uint8_t command, sysexCallbackFunction newFunction, void ∗context=NULL)
- void attach (uint8_t command, systemCallbackFunction newFunction, void ∗context=NULL)
- void attach (uint8_t command, versionCallbackFunction newFunction, void ∗context=NULL)
- void detach (uint8_t command)
- void detach (dataBufferOverflowCallbackFunction)

## 26.8.1 Detailed Description

Definition at line 27 of file FirmataParser.h.

## 26.8.2 Member Typedef Documentation

### 26.8.2.1 callbackFunction

```
typedef void(* firmata::FirmataParser::callbackFunction) (void *context, uint8_t command,
uint16_t value)
```

Definition at line 31 of file FirmataParser.h.

### 26.8.2.2 dataBufferOverflowCallbackFunction

```
typedef void(* firmata::FirmataParser::dataBufferOverflowCallbackFunction) (void *context)
```

Definition at line 32 of file FirmataParser.h.

#### 26.8.2.3 stringCallbackFunction

```
typedef void(* firmata::FirmataParser::stringCallbackFunction) (void *context, const char *c_↩
str)
```

Definition at line 33 of file FirmataParser.h.

#### 26.8.2.4 sysexCallbackFunction

```
typedef void(* firmata::FirmataParser::sysexCallbackFunction) (void *context, uint8_t command,
size_t argc, uint8_t *argv)
```

Definition at line 34 of file FirmataParser.h.

#### 26.8.2.5 systemCallbackFunction

```
typedef void(* firmata::FirmataParser::systemCallbackFunction) (void *context)
```

Definition at line 35 of file FirmataParser.h.

#### 26.8.2.6 versionCallbackFunction

```
typedef void(* firmata::FirmataParser::versionCallbackFunction) (void *context, size_t sv_↩
major, size_t sv_minor, const char *firmware)
```

Definition at line 36 of file FirmataParser.h.

### 26.8.3 Constructor & Destructor Documentation

#### 26.8.3.1 FirmataParser()

```
FirmataParser::FirmataParser (
            uint8_t * dataBuffer = (uint8_t *)NULL,
            size_t dataBufferSize = 0 )
```

The FirmataParser class.

**Parameters**

| | |
|---|---|
| *dataBuffer* | A pointer to an external buffer used to store parsed data |
| *dataBufferSize* | The size of the external buffer |

Definition at line 33 of file FirmataParser.cpp.

```
:
  dataBuffer(dataBuffer),
  dataBufferSize(dataBufferSize),
  executeMultiByteCommand(0),
  multiByteChannel(0),
  waitForData(0),
  parsingSysex(false),
  sysexBytesRead(0),
  currentAnalogCallbackContext((void *)NULL),
  currentDigitalCallbackContext((void *)NULL),
  currentReportAnalogCallbackContext((void *)NULL),
  currentReportDigitalCallbackContext((void *)NULL),
  currentPinModeCallbackContext((void *)NULL),
  currentPinValueCallbackContext((void *)NULL),
  currentReportFirmwareCallbackContext((void *)NULL),
  currentReportVersionCallbackContext((void *)NULL),
  currentDataBufferOverflowCallbackContext((void *)NULL),
  currentStringCallbackContext((void *)NULL),
  currentSysexCallbackContext((void *)NULL),
  currentSystemResetCallbackContext((void *)NULL),
  currentAnalogCallback((callbackFunction)NULL),
  currentDigitalCallback((callbackFunction)NULL),
  currentReportAnalogCallback((callbackFunction)NULL),
  currentReportDigitalCallback((callbackFunction)NULL),
  currentPinModeCallback((callbackFunction)NULL),
  currentPinValueCallback((callbackFunction)NULL),
  currentDataBufferOverflowCallback((dataBufferOverflowCallbackFunction)NULL),
  currentStringCallback((stringCallbackFunction)NULL),
  currentSysexCallback((sysexCallbackFunction)NULL),
  currentReportFirmwareCallback((versionCallbackFunction)NULL),
  currentReportVersionCallback((systemCallbackFunction)NULL),
  currentSystemResetCallback((systemCallbackFunction)NULL)
{
    allowBufferUpdate = ((uint8_t *)NULL == dataBuffer);
}
```

### 26.8.4 Member Function Documentation

#### 26.8.4.1 attach() [1/6]

```
void FirmataParser::attach (
            dataBufferOverflowCallbackFunction newFunction,
            void * context = NULL )
```

Attach a buffer overflow callback

**Parameters**

| | |
|---|---|
| *newFunction* | A reference to the buffer overflow callback function to attach. |
| *context* | An optional context to be provided to the callback function (NULL by default). |

**Note**

The context parameter is provided so you can pass a parameter, by reference, to your callback function.

Definition at line 326 of file FirmataParser.cpp.

```
{
  currentDataBufferOverflowCallback = newFunction;
  currentDataBufferOverflowCallbackContext = context;
}
```

#### 26.8.4.2 attach() [2/6]

```
void FirmataParser::attach (
            uint8_t command,
            callbackFunction newFunction,
            void * context = NULL )
```

Attach a generic sysex callback function to a command (options are: ANALOG_MESSAGE, DIGITAL_MESSAGE, REPORT_ANALOG, REPORT DIGITAL, SET_PIN_MODE and SET_DIGITAL_PIN_VALUE).

**Parameters**

| | |
|---|---|
| *command* | The ID of the command to attach a callback function to. |
| *newFunction* | A reference to the callback function to attach. |
| *context* | An optional context to be provided to the callback function (NULL by default). |

**Note**

The context parameter is provided so you can pass a parameter, by reference, to your callback function.

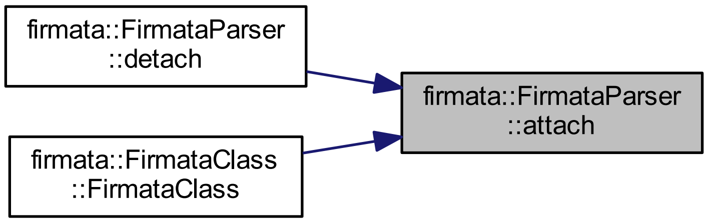
Definition at line 216 of file FirmataParser.cpp.

```
{
  switch (command) {
    case ANALOG_MESSAGE:
      currentAnalogCallback = newFunction;
      currentAnalogCallbackContext = context;
      break;
    case DIGITAL_MESSAGE:
      currentDigitalCallback = newFunction;
      currentDigitalCallbackContext = context;
      break;
    case REPORT_ANALOG:
      currentReportAnalogCallback = newFunction;
      currentReportAnalogCallbackContext = context;
      break;
    case REPORT_DIGITAL:
      currentReportDigitalCallback = newFunction;
      currentReportDigitalCallbackContext = context;
      break;
    case SET_PIN_MODE:
      currentPinModeCallback = newFunction;
      currentPinModeCallbackContext = context;
      break;
    case SET_DIGITAL_PIN_VALUE:
      currentPinValueCallback = newFunction;
      currentPinValueCallbackContext = context;
      break;
  }
}
```

References firmata::ANALOG_MESSAGE, firmata::DIGITAL_MESSAGE, firmata::REPORT_ANALOG, firmata::↩
REPORT_DIGITAL, firmata::SET_DIGITAL_PIN_VALUE, and firmata::SET_PIN_MODE.

Referenced by detach(), and firmata::FirmataClass::FirmataClass().

Here is the caller graph for this function:


firmata::FirmataParser ::detach
firmata::FirmataClass ::FirmataClass
firmata::FirmataParser ::attach

### 26.8.4.3 attach() [3/6]

```
void FirmataParser::attach (
             uint8_t command,
             stringCallbackFunction newFunction,
             void * context = NULL )
```

Attach a callback function for the STRING_DATA command.

**Parameters**

| | |
|---|---|
| *command* | Must be set to STRING_DATA or it will be ignored. |
| *newFunction* | A reference to the string callback function to attach. |
| *context* | An optional context to be provided to the callback function (NULL by default). |

**Note**

The context parameter is provided so you can pass a parameter, by reference, to your callback function.

Definition at line 294 of file FirmataParser.cpp.

```
{
  switch (command) {
    case STRING_DATA:
      currentStringCallback = newFunction;
      currentStringCallbackContext = context;
      break;
  }
}
```

References firmata::STRING_DATA.

#### 26.8.4.4 attach() [4/6]

```
void FirmataParser::attach (
            uint8_t command,
            sysexCallbackFunction newFunction,
            void * context = NULL )
```

Attach a generic sysex callback function to sysex command.

**Parameters**

| | |
|---|---|
| *command* | The ID of the command to attach a callback function to. |
| *newFunction* | A reference to the sysex callback function to attach. |
| *context* | An optional context to be provided to the callback function (NULL by default). |

**Note**

The context parameter is provided so you can pass a parameter, by reference, to your callback function.

Definition at line 312 of file FirmataParser.cpp.

```
{
  (void)command;
  currentSysexCallback = newFunction;
  currentSysexCallbackContext = context;
}
```

#### 26.8.4.5 attach() [5/6]

```
void FirmataParser::attach (
            uint8_t command,
            systemCallbackFunction newFunction,
            void * context = NULL )
```

Attach a system callback function (supported options are: SYSTEM_RESET, REPORT_VERSION).

**Parameters**

| | |
|---|---|
| *command* | The ID of the command to attach a callback function to. |
| *newFunction* | A reference to the callback function to attach. |
| *context* | An optional context to be provided to the callback function (NULL by default). |

**Note**

The context parameter is provided so you can pass a parameter, by reference, to your callback function.

Definition at line 272 of file FirmataParser.cpp.

```
{
  switch (command) {
    case REPORT_VERSION:
      currentReportVersionCallback = newFunction;
      currentReportVersionCallbackContext = context;
      break;
    case SYSTEM_RESET:
      currentSystemResetCallback = newFunction;
      currentSystemResetCallbackContext = context;
      break;
  }
}
```

References firmata::REPORT_VERSION, and firmata::SYSTEM_RESET.

#### 26.8.4.6 attach() [6/6]

```
void FirmataParser::attach (
            uint8_t command,
            versionCallbackFunction newFunction,
            void * context = NULL )
```

Attach a version callback function (supported option: REPORT_FIRMWARE).

**Parameters**

| | |
|---|---|
| *command* | The ID of the command to attach a callback function to. |
| *newFunction* | A reference to the callback function to attach. |
| *context* | An optional context to be provided to the callback function (NULL by default). |

**Note**

The context parameter is provided so you can pass a parameter, by reference, to your callback function.

Definition at line 254 of file FirmataParser.cpp.

```
{
  switch (command) {
    case REPORT_FIRMWARE:
      currentReportFirmwareCallback = newFunction;
      currentReportFirmwareCallbackContext = context;
      break;
  }
}
```

References firmata::REPORT_FIRMWARE.

#### 26.8.4.7 detach() [1/2]

```
void FirmataParser::detach (
            dataBufferOverflowCallbackFunction  )
```

Detach the buffer overflow callback

**Parameters**

| | |
|---|---|
| *<unused>* | Any pointer of type dataBufferOverflowCallbackFunction. |

Definition at line 363 of file FirmataParser.cpp.

```
{
  currentDataBufferOverflowCallback = (dataBufferOverflowCallbackFunction)NULL;
  currentDataBufferOverflowCallbackContext = (void *)NULL;
}
```

### 26.8.4.8 detach() [2/2]

```
void FirmataParser::detach (
            uint8_t command )
```

Detach a callback function for a specified command (such as SYSTEM_RESET, STRING_DATA, ANALOG_ME←SSAGE, DIGITAL_MESSAGE, etc).

**Parameters**

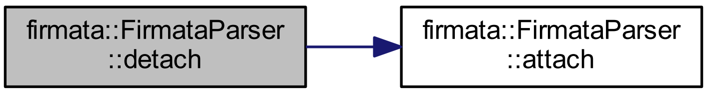
| | |
|---|---|
| *command* | The ID of the command to detatch the callback function from. |

Definition at line 337 of file FirmataParser.cpp.

```
{
  switch (command) {
    case REPORT_FIRMWARE:
      attach(command, (versionCallbackFunction)NULL, NULL);
      break;
    case REPORT_VERSION:
    case SYSTEM_RESET:
      attach(command, (systemCallbackFunction)NULL, NULL);
      break;
    case STRING_DATA:
      attach(command, (stringCallbackFunction)NULL, NULL);
      break;
    case START_SYSEX:
      attach(command, (sysexCallbackFunction)NULL, NULL);
      break;
    default:
      attach(command, (callbackFunction)NULL, NULL);
      break;
  }
}
```

References attach(), firmata::REPORT_FIRMWARE, firmata::REPORT_VERSION, firmata::START_SYSEX, firmata::STRING_DATA, and firmata::SYSTEM_RESET.

Here is the call graph for this function:


firmata::FirmataParser ::detach
firmata::FirmataParser ::attach

### 26.8.4.9 isParsingMessage()

```
bool FirmataParser::isParsingMessage (
             void  ) const
```

**Returns**

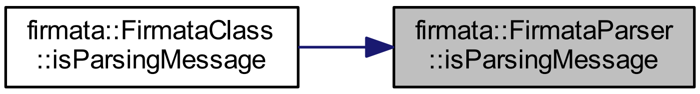
Returns true if the parser is actively parsing data.

Definition at line 176 of file FirmataParser.cpp.

```
{
  return (waitForData > 0 || parsingSysex);
}
```

Referenced by firmata::FirmataClass::isParsingMessage().

Here is the caller graph for this function:


firmata::FirmataClass ::isParsingMessage
firmata::FirmataParser ::isParsingMessage

### 26.8.4.10 parse()

```
void FirmataParser::parse (
             uint8_t inputData )
```

Parse data from the input stream.

**Parameters**

| | |
|---|---|
| *inputData* | A single byte to be added to the parser. |

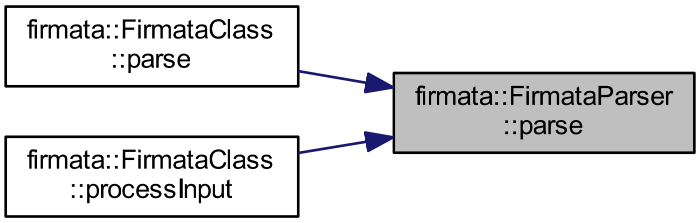
Definition at line 81 of file FirmataParser.cpp.

```
{
  uint8_t command;

  if (parsingSysex) {
    if (inputData == END_SYSEX) {
      //stop sysex byte
      parsingSysex = false;
      //fire off handler function
      processSysexMessage();
    } else {
      //normal data byte - add to buffer
      bufferDataAtPosition(inputData, sysexBytesRead);
      ++sysexBytesRead;
    }
  } else if ( (waitForData > 0) && (inputData < 128) ) {
    --waitForData;
    bufferDataAtPosition(inputData, waitForData);
    if ( (waitForData == 0) && executeMultiByteCommand ) { // got the whole message
      switch (executeMultiByteCommand) {
        case ANALOG_MESSAGE:
          if (currentAnalogCallback) {
            (*currentAnalogCallback)(currentAnalogCallbackContext,
                                     multiByteChannel,
                                     (dataBuffer[0] « 7)
                                     + dataBuffer[1]);
          }
          break;
        case DIGITAL_MESSAGE:
          if (currentDigitalCallback) {
            (*currentDigitalCallback)(currentDigitalCallbackContext,
                                      multiByteChannel,
                                      (dataBuffer[0] « 7)
                                      + dataBuffer[1]);
          }
          break;
        case SET_PIN_MODE:
          if (currentPinModeCallback)
            (*currentPinModeCallback)(currentPinModeCallbackContext, dataBuffer[1], dataBuffer[0]);
          break;
        case SET_DIGITAL_PIN_VALUE:
          if (currentPinValueCallback)
            (*currentPinValueCallback)(currentPinValueCallbackContext, dataBuffer[1], dataBuffer[0]);
          break;
        case REPORT_ANALOG:
          if (currentReportAnalogCallback)
            (*currentReportAnalogCallback)(currentReportAnalogCallbackContext, multiByteChannel,
    dataBuffer[0]);
          break;
        case REPORT_DIGITAL:
          if (currentReportDigitalCallback)
            (*currentReportDigitalCallback)(currentReportDigitalCallbackContext, multiByteChannel,
    dataBuffer[0]);
          break;
      }
      executeMultiByteCommand = 0;
    }
  } else {
    // remove channel info from command byte if less than 0xF0
    if (inputData < 0xF0) {
      command = inputData & 0xF0;
      multiByteChannel = inputData & 0x0F;
    } else {
      command = inputData;
      // commands in the 0xF* range don't use channel data
    }
    switch (command) {
      case ANALOG_MESSAGE:
      case DIGITAL_MESSAGE:
      case SET_PIN_MODE:
      case SET_DIGITAL_PIN_VALUE:
        waitForData = 2; // two data bytes needed
        executeMultiByteCommand = command;
        break;
      case REPORT_ANALOG:
      case REPORT_DIGITAL:
        waitForData = 1; // one data byte needed
        executeMultiByteCommand = command;
        break;
      case START_SYSEX:
        parsingSysex = true;
        sysexBytesRead = 0;
        break;
      case SYSTEM_RESET:
        systemReset();
        break;
      case REPORT_VERSION:
        if (currentReportVersionCallback)
          (*currentReportVersionCallback)(currentReportVersionCallbackContext);
        break;
    }
  }
}
```

References firmata::ANALOG_MESSAGE, firmata::DIGITAL_MESSAGE, firmata::END_SYSEX, firmata::REPORT_ANALOG, firmata::REPORT_DIGITAL, firmata::REPORT_VERSION, firmata::SET_DIGITAL_PIN_VALUE, firmata::SET_PIN_MODE, firmata::START_SYSEX, and firmata::SYSTEM_RESET.

Referenced by firmata::FirmataClass::parse(), and firmata::FirmataClass::processInput().

Here is the caller graph for this function:


firmata::FirmataClass ::parse
firmata::FirmataClass ::processInput
firmata::FirmataParser ::parse

### 26.8.4.11 setDataBufferOfSize()

```
int FirmataParser::setDataBufferOfSize (
             uint8_t * dataBuffer,
             size_t dataBufferSize )
```

Provides a mechanism to either set or update the working buffer of the parser. The method will be enabled when no buffer has been provided, or an overflow condition exists.

**Parameters**

| | |
|---|---|
| *dataBuffer* | A pointer to an external buffer used to store parsed data |
| *dataBufferSize* | The size of the external buffer |

Definition at line 189 of file FirmataParser.cpp.

```
{
    int result;

    if ( !allowBufferUpdate ) {
      result = __LINE__;
    } else if ((uint8_t *)NULL == dataBuffer) {
      result = __LINE__;
    } else {
      this->dataBuffer = dataBuffer;
      this->dataBufferSize = dataBufferSize;
      allowBufferUpdate = false;
      result = 0;
    }

    return result;
}
```

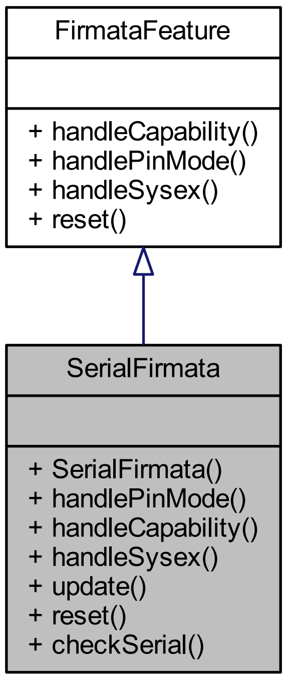
The documentation for this class was generated from the following files:

- FirmataParser.h
- FirmataParser.cpp

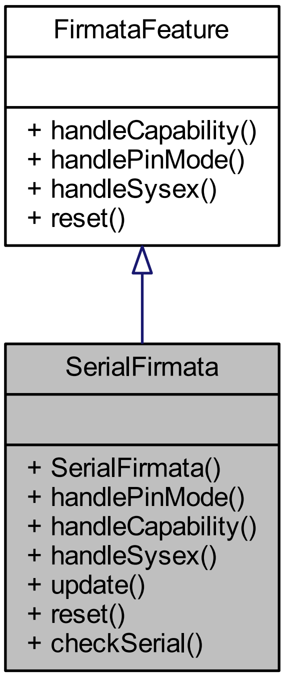
## 26.9 SerialFirmata Class Reference

```
#include <SerialFirmata.h>
```

Inheritance diagram for SerialFirmata:



Collaboration diagram for SerialFirmata:



## Public Member Functions

- SerialFirmata ()
- boolean handlePinMode (byte pin, int mode)
- void handleCapability (byte pin)
- boolean handleSysex (byte command, byte argc, byte ∗argv)
- void update ()
- void reset ()
- void checkSerial ()

### 26.9.1 Detailed Description

Definition at line 181 of file SerialFirmata.h.

### 26.9.2 Constructor & Destructor Documentation

#### 26.9.2.1 SerialFirmata()

```
SerialFirmata::SerialFirmata ( )
```

Definition at line 22 of file SerialFirmata.cpp.

```
{
#if defined(SoftwareSerial_h)
  swSerial0 = NULL;
  swSerial1 = NULL;
  swSerial2 = NULL;
  swSerial3 = NULL;
#endif

  serialIndex = -1;
}
```

### 26.9.3 Member Function Documentation

#### 26.9.3.1 checkSerial()

```
void SerialFirmata::checkSerial ( )
```

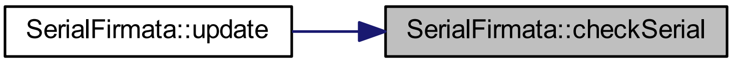
Definition at line 296 of file SerialFirmata.cpp.

```
{
  byte portId, serialData;
  int bytesToRead = 0;
  int numBytesToRead = 0;
  Stream* serialPort;

  if (serialIndex > -1) {

    // loop through all reporting (READ_CONTINUOUS) serial ports
    for (byte i = 0; i < serialIndex + 1; i++) {
      portId = reportSerial[i];
      bytesToRead = serialBytesToRead[portId];
      serialPort = getPortFromId(portId);
      if (serialPort == NULL) {
        continue;
      }
#if defined(SoftwareSerial_h)
      // only the SoftwareSerial port that is "listening" can read data
      if (portId > 7 && !((SoftwareSerial*)serialPort)->isListening()) {
        continue;
      }
#endif
      if (serialPort->available() > 0) {
        Firmata.write(START_SYSEX);
        Firmata.write(SERIAL_MESSAGE);
        Firmata.write(SERIAL_REPLY | portId);

        if (bytesToRead == 0 || (serialPort->available() <= bytesToRead)) {
          numBytesToRead = serialPort->available();
        } else {
          numBytesToRead = bytesToRead;
        }

        // relay serial data to the serial device
        while (numBytesToRead > 0) {
          serialData = serialPort->read();
          Firmata.write(serialData & 0x7F);
          Firmata.write((serialData » 7) & 0x7F);
          numBytesToRead--;
        }
        Firmata.write(END_SYSEX);
      }

    }
  }
}
```

References END_SYSEX, Firmata, SERIAL_MESSAGE, SERIAL_REPLY, and START_SYSEX.

Referenced by update().

Here is the caller graph for this function:



### 26.9.3.2 handleCapability()

```
void SerialFirmata::handleCapability (
             byte pin )  [virtual]
```

Implements FirmataFeature.

Definition at line 44 of file SerialFirmata.cpp.

```
{
  if (IS_PIN_SERIAL(pin)) {
    Firmata.write(PIN_MODE_SERIAL);
    Firmata.write(getSerialPinType(pin));
  }
}
```

References Firmata, IS_PIN_SERIAL, and PIN_MODE_SERIAL.

### 26.9.3.3 handlePinMode()

```
boolean SerialFirmata::handlePinMode (
             byte pin,
             int mode )  [virtual]
```

Implements FirmataFeature.

Definition at line 34 of file SerialFirmata.cpp.

```
{
  // used for both HW and SW serial
  if (mode == PIN_MODE_SERIAL) {
    Firmata.setPinMode(pin, PIN_MODE_SERIAL);
    return true;
  }
  return false;
}
```

References Firmata, and PIN_MODE_SERIAL.

### 26.9.3.4 handleSysex()

```
boolean SerialFirmata::handleSysex (
        byte command,
        byte argc,
        byte * argv )  [virtual]
```

Implements FirmataFeature.

Definition at line 52 of file SerialFirmata.cpp.

```
{
  if (command == SERIAL_MESSAGE) {

    Stream *serialPort;
    byte mode = argv[0] & SERIAL_MODE_MASK;
    byte portId = argv[0] & SERIAL_PORT_ID_MASK;

    switch (mode) {
      case SERIAL_CONFIG:
        {
          long baud = (long)argv[1] | ((long)argv[2] << 7) | ((long)argv[3] << 14);
          serial_pins pins;

          if (portId < 8) {
            serialPort = getPortFromId(portId);
            if (serialPort != NULL) {
              pins = getSerialPinNumbers(portId);
              if (pins.rx != 0 && pins.tx != 0) {
                Firmata.setPinMode(pins.rx, PIN_MODE_SERIAL);
                Firmata.setPinMode(pins.tx, PIN_MODE_SERIAL);
                // Fixes an issue where some serial devices would not work properly with Arduino Due
                // because all Arduino pins are set to OUTPUT by default in StandardFirmata.
                pinMode(pins.rx, INPUT);
              }
              ((HardwareSerial*)serialPort)->begin(baud);
            }
          } else {
#if defined(SoftwareSerial_h)
            byte swTxPin, swRxPin;
            if (argc > 4) {
              swRxPin = argv[4];
              swTxPin = argv[5];
            } else {
              // RX and TX pins must be specified when using SW serial
              Firmata.sendString("Specify serial RX and TX pins");
              return false;
            }
            switch (portId) {
              case SW_SERIAL0:
                if (swSerial0 == NULL) {
                  swSerial0 = new SoftwareSerial(swRxPin, swTxPin);
                }
                break;
              case SW_SERIAL1:
                if (swSerial1 == NULL) {
                  swSerial1 = new SoftwareSerial(swRxPin, swTxPin);
                }
                break;
              case SW_SERIAL2:
                if (swSerial2 == NULL) {
                  swSerial2 = new SoftwareSerial(swRxPin, swTxPin);
                }
                break;
              case SW_SERIAL3:
                if (swSerial3 == NULL) {
                  swSerial3 = new SoftwareSerial(swRxPin, swTxPin);
                }
                break;
            }
            serialPort = getPortFromId(portId);
            if (serialPort != NULL) {
              Firmata.setPinMode(swRxPin, PIN_MODE_SERIAL);
              Firmata.setPinMode(swTxPin, PIN_MODE_SERIAL);
              ((SoftwareSerial*)serialPort)->begin(baud);
            }
#endif
          }
          break; // SERIAL_CONFIG
        }
      case SERIAL_WRITE:
        {
          byte data;
          serialPort = getPortFromId(portId);
          if (serialPort == NULL) {
            break;
          }
          for (byte i = 1; i < argc; i += 2) {
            data = argv[i] + (argv[i + 1] « 7);
            serialPort->write(data);
          }
          break; // SERIAL_WRITE
        }
      case SERIAL_READ:
        if (argv[1] == SERIAL_READ_CONTINUOUSLY) {
          if (serialIndex + 1 >= MAX_SERIAL_PORTS) {
            break;
          }

          if (argc > 2) {
            // maximum number of bytes to read from buffer per iteration of loop()
            serialBytesToRead[portId] = (int)argv[2] | ((int)argv[3] « 7);
          } else {
            // read all available bytes per iteration of loop()
            serialBytesToRead[portId] = 0;
          }
          serialIndex++;
          reportSerial[serialIndex] = portId;
        } else if (argv[1] == SERIAL_STOP_READING) {
          byte serialIndexToSkip = 0;
          if (serialIndex <= 0) {
            serialIndex = -1;
          } else {
            for (byte i = 0; i < serialIndex + 1; i++) {
              if (reportSerial[i] == portId) {
                serialIndexToSkip = i;
                break;
              }
            }
            // shift elements over to fill space left by removed element
            for (byte i = serialIndexToSkip; i < serialIndex + 1; i++) {
              if (i < MAX_SERIAL_PORTS) {
                reportSerial[i] = reportSerial[i + 1];
              }
            }
            serialIndex--;
          }
        }
        break; // SERIAL_READ
      case SERIAL_CLOSE:
        serialPort = getPortFromId(portId);
        if (serialPort != NULL) {
          if (portId < 8) {
            ((HardwareSerial*)serialPort)->end();
          } else {
#if defined(SoftwareSerial_h)
            ((SoftwareSerial*)serialPort)->end();
            if (serialPort != NULL) {
              free(serialPort);
              serialPort = NULL;
            }
#endif
          }
        }
        break; // SERIAL_CLOSE
      case SERIAL_FLUSH:
        serialPort = getPortFromId(portId);
        if (serialPort != NULL) {
          getPortFromId(portId)->flush();
        }
        break; // SERIAL_FLUSH
#if defined(SoftwareSerial_h)
      case SERIAL_LISTEN:
        // can only call listen() on software serial ports
        if (portId > 7) {
          serialPort = getPortFromId(portId);
          if (serialPort != NULL) {
            ((SoftwareSerial*)serialPort)->listen();
          }
        }
        break; // SERIAL_LISTEN
#endif
    } // end switch
    return true;
  }
  return false;
}
```

References Firmata, MAX_SERIAL_PORTS, PIN_MODE_SERIAL, SERIAL_CLOSE, SERIAL_CONFIG, SERIA↩
L_FLUSH, SERIAL_LISTEN, SERIAL_MESSAGE, SERIAL_MODE_MASK, SERIAL_PORT_ID_MASK, SERIAL↩

_READ, SERIAL_READ_CONTINUOUSLY, SERIAL_STOP_READING, SERIAL_WRITE, SW_SERIAL0, SW_S↩
ERIAL1, SW_SERIAL2, and SW_SERIAL3.

#### 26.9.3.5 reset()

```
void SerialFirmata::reset ( )  [virtual]
```

Implements FirmataFeature.

Definition at line 215 of file SerialFirmata.cpp.

```
{
#if defined(SoftwareSerial_h)
  Stream *serialPort;
  // free memory allocated for SoftwareSerial ports
  for (byte i = SW_SERIAL0; i < SW_SERIAL3 + 1; i++) {
    serialPort = getPortFromId(i);
    if (serialPort != NULL) {
      free(serialPort);
      serialPort = NULL;
    }
  }
#endif

  serialIndex = -1;
  for (byte i = 0; i < SERIAL_READ_ARR_LEN; i++) {
    serialBytesToRead[i] = 0;
  }
}
```

References SERIAL_READ_ARR_LEN, SW_SERIAL0, and SW_SERIAL3.

#### 26.9.3.6 update()

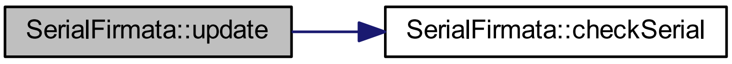
```
void SerialFirmata::update ( )
```

Definition at line 210 of file SerialFirmata.cpp.

```
{
  checkSerial();
}
```

References checkSerial().

Here is the call graph for this function:



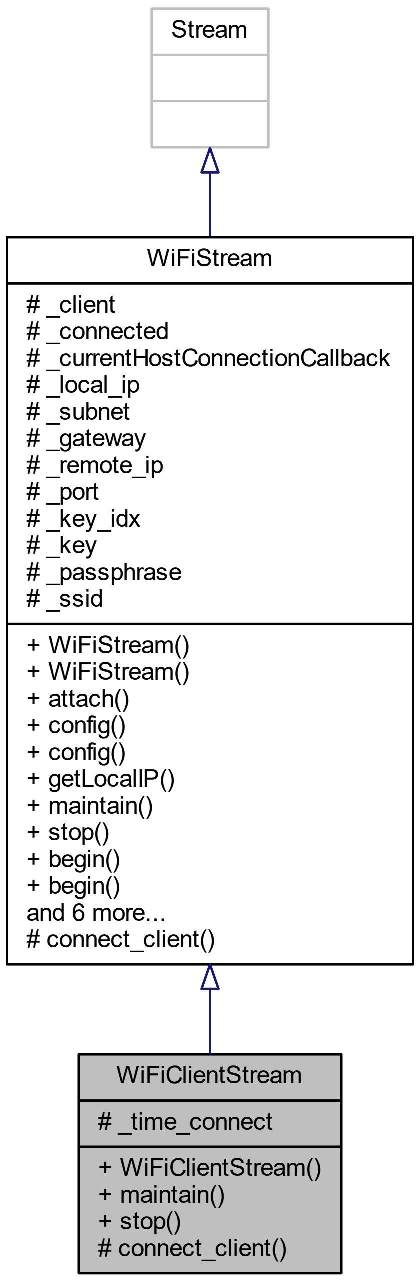
The documentation for this class was generated from the following files:

- SerialFirmata.h
- SerialFirmata.cpp

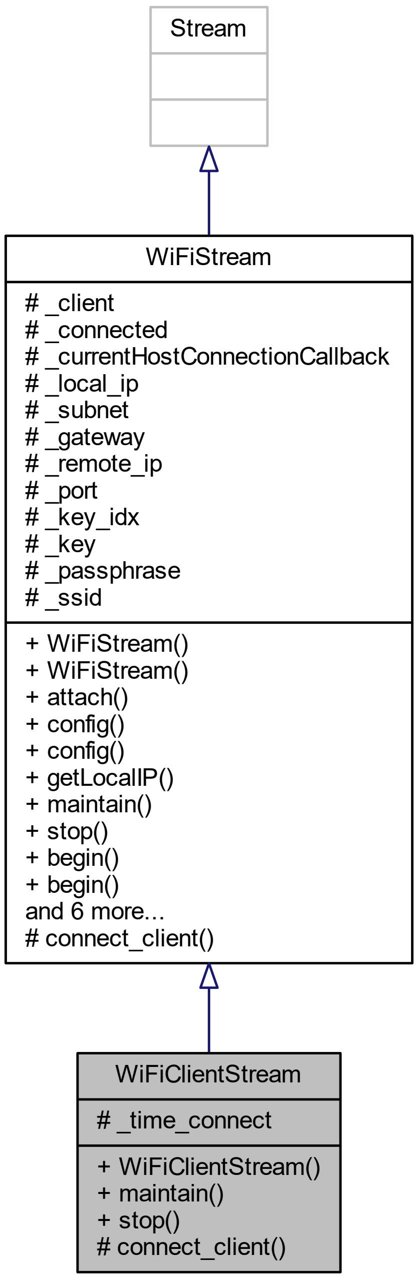
## 26.10 WiFiClientStream Class Reference

```
#include <WiFiClientStream.h>
```

Inheritance diagram for WiFiClientStream:



Collaboration diagram for WiFiClientStream:



## Public Member Functions

- WiFiClientStream (IPAddress server_ip, uint16_t server_port)
- virtual bool maintain ()
- virtual void stop ()

## Protected Member Functions

- virtual bool connect_client ()

## Protected Attributes

- uint32_t _time_connect = 0

### 26.10.1 Detailed Description

Definition at line 33 of file WiFiClientStream.h.

### 26.10.2 Constructor & Destructor Documentation

#### 26.10.2.1 WiFiClientStream()

```
WiFiClientStream::WiFiClientStream (
            IPAddress server_ip,
            uint16_t server_port ) [inline]
```

create a WiFi stream with a TCP client

Definition at line 75 of file WiFiClientStream.h.

```
75 : WiFiStream(server_ip, server_port) {}
```

### 26.10.3 Member Function Documentation

#### 26.10.3.1 connect_client()

```
virtual bool WiFiClientStream::connect_client ( ) [inline], [protected], [virtual]
```

check if TCP client is connected

**Returns**

true if connected

Implements WiFiStream.

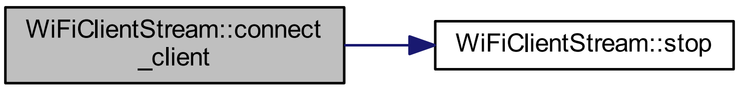
Definition at line 42 of file WiFiClientStream.h.

```
{
  if ( _connected )
  {
    if ( _client && _client.connected() ) return true;
    stop();
  }

  // active TCP connect
  if ( WiFi.status() == WL_CONNECTED )
  {
    // if the client is disconnected, try to reconnect every 5 seconds
    if ( millis() - _time_connect >= MILLIS_RECONNECT )
    {
      _connected = _client.connect( _remote_ip, _port );
      if ( !_connected )
      {
        _time_connect = millis();
      }
      else if ( _currentHostConnectionCallback )
      {
        (*_currentHostConnectionCallback)(HOST_CONNECTION_CONNECTED);
      }
    }
  }

  return _connected;
}
```

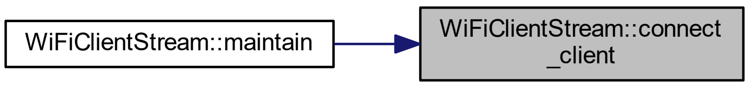
References WiFiStream::_client, WiFiStream::_connected, WiFiStream::_currentHostConnectionCallback, WiFi↵ Stream::_port, WiFiStream::_remote_ip, _time_connect, HOST_CONNECTION_CONNECTED, MILLIS_RECO↵ NNECT, and stop().

Referenced by maintain().

Here is the call graph for this function:



Here is the caller graph for this function:



### 26.10.3.2 maintain()

```
virtual bool WiFiClientStream::maintain ( )  [inline], [virtual]
```

maintain WiFi and TCP connection

**Returns**

true if WiFi and TCP connection are established

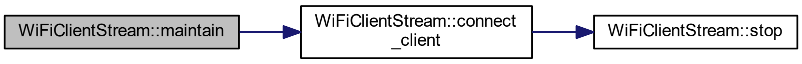
Implements WiFiStream.

Definition at line 81 of file WiFiClientStream.h.

```
{
  return connect_client();
}
```

References connect_client().

Here is the call graph for this function:



### 26.10.3.3 stop()

```
virtual void WiFiClientStream::stop ( )  [inline], [virtual]
```

stop client connection

Implements WiFiStream.

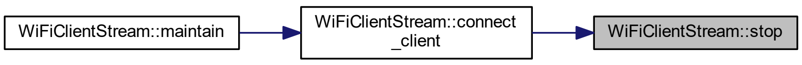
Definition at line 89 of file WiFiClientStream.h.

```
{
  if ( _client)
  {
    _client.stop();
    if ( _currentHostConnectionCallback )
    {
      (*_currentHostConnectionCallback)(HOST_CONNECTION_DISCONNECTED);
    }
  }
  _connected = false;
  _time_connect = millis();
}
```

References WiFiStream::_client, WiFiStream::_connected, WiFiStream::_currentHostConnectionCallback, _time↩_connect, and HOST_CONNECTION_DISCONNECTED.

Referenced by connect_client().

Here is the caller graph for this function:



### 26.10.4 Field Documentation

#### 26.10.4.1 _time_connect

```
uint32_t WiFiClientStream::_time_connect = 0  [protected]
```

Definition at line 36 of file WiFiClientStream.h.

Referenced by connect_client(), and stop().

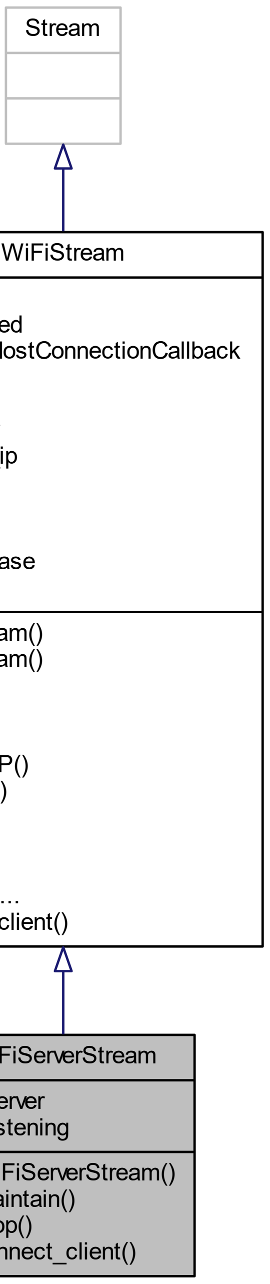
The documentation for this class was generated from the following file:

- WiFiClientStream.h

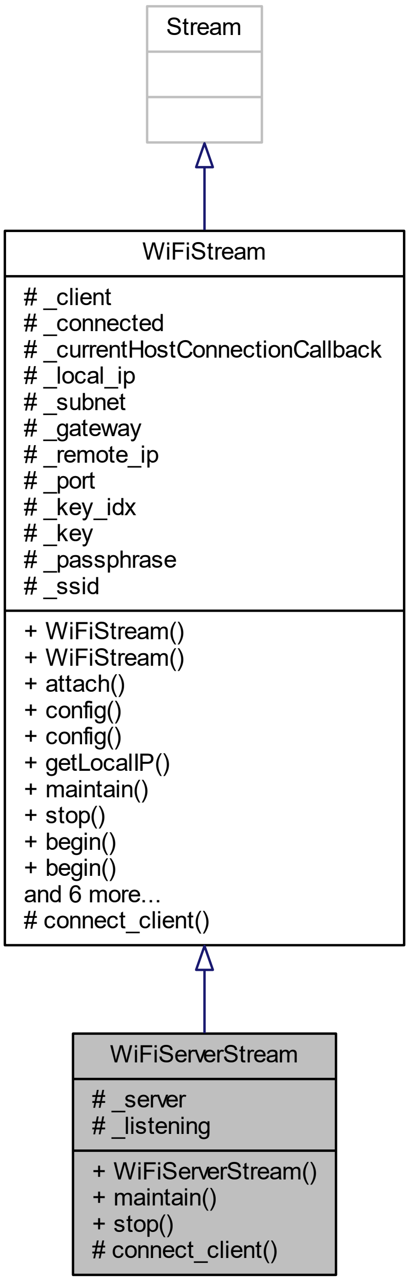
## 26.11 WiFiServerStream Class Reference

```
#include <WiFiServerStream.h>
```

Inheritance diagram for WiFiServerStream:



Collaboration diagram for WiFiServerStream:



## Public Member Functions

- WiFiServerStream (uint16_t server_port)
- virtual bool maintain ()
- virtual void stop ()

## Protected Member Functions

- virtual bool connect_client ()

## Protected Attributes

- WiFiServer _server = WiFiServer(3030)
- bool _listening = false

### 26.11.1 Detailed Description

Definition at line 31 of file WiFiServerStream.h.

### 26.11.2 Constructor & Destructor Documentation

#### 26.11.2.1 WiFiServerStream()

```
WiFiServerStream::WiFiServerStream (
            uint16_t server_port )  [inline]
```

create a WiFi stream with a TCP server

Definition at line 66 of file WiFiServerStream.h.

```
66 : WiFiStream(server_port) {}
```

### 26.11.3 Member Function Documentation

#### 26.11.3.1 connect_client()

```
virtual bool WiFiServerStream::connect_client ( )  [inline], [protected], [virtual]
```

check if TCP client is connected

**Returns**

true if connected

Implements WiFiStream.

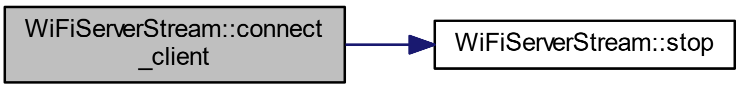
Definition at line 41 of file WiFiServerStream.h.

```
{
  if ( _connected )
  {
    if ( _client && _client.connected() ) return true;
    stop();
  }

  // passive TCP connect (accept)
  WiFiClient newClient = _server.available();
  if ( !newClient ) return false;
  _client = newClient;
  _connected = true;
  if ( _currentHostConnectionCallback )
  {
    (*_currentHostConnectionCallback)(HOST_CONNECTION_CONNECTED);
  }

  return true;
}
```

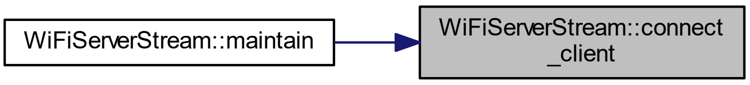
References WiFiStream::_client, WiFiStream::_connected, WiFiStream::_currentHostConnectionCallback, _server, HOST_CONNECTION_CONNECTED, and stop().

Referenced by maintain().

Here is the call graph for this function:



Here is the caller graph for this function:



### 26.11.3.2 maintain()

```
virtual bool WiFiServerStream::maintain ( )  [inline], [virtual]
```

maintain WiFi and TCP connection

**Returns**

true if WiFi and TCP connection are established

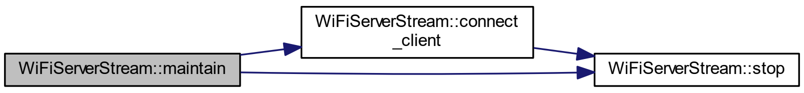
Implements WiFiStream.

Definition at line 72 of file WiFiServerStream.h.

```
{
  if ( connect_client() ) return true;

  stop();

  if ( !_listening && WiFi.status() == WL_CONNECTED )
  {
    // start TCP server after first WiFi connect
    _server = WiFiServer(_port);
    _server.begin();
    _listening = true;
  }

  return false;
}
```

References _listening, WiFiStream::_port, _server, connect_client(), and stop().

Here is the call graph for this function:



### 26.11.3.3 stop()

```
virtual void WiFiServerStream::stop ( )  [inline], [virtual]
```

stop client connection

Implements WiFiStream.

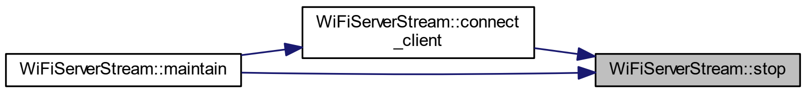
Definition at line 92 of file WiFiServerStream.h.

```
{
  if ( _client)
  {
    _client.stop();
    if ( _currentHostConnectionCallback )
    {
      (*_currentHostConnectionCallback)(HOST_CONNECTION_DISCONNECTED);
    }
  }
  _connected = false;
}
```

References WiFiStream::_client, WiFiStream::_connected, WiFiStream::_currentHostConnectionCallback, and H↩
OST_CONNECTION_DISCONNECTED.

Referenced by connect_client(), and maintain().

Here is the caller graph for this function:



## 26.11.4 Field Documentation

### 26.11.4.1 _listening

```
bool WiFiServerStream::_listening = false  [protected]
```

Definition at line 35 of file WiFiServerStream.h.

Referenced by maintain().

### 26.11.4.2 _server

```
WiFiServer WiFiServerStream::_server = WiFiServer(3030)  [protected]
```

Definition at line 34 of file WiFiServerStream.h.

Referenced by connect_client(), and maintain().

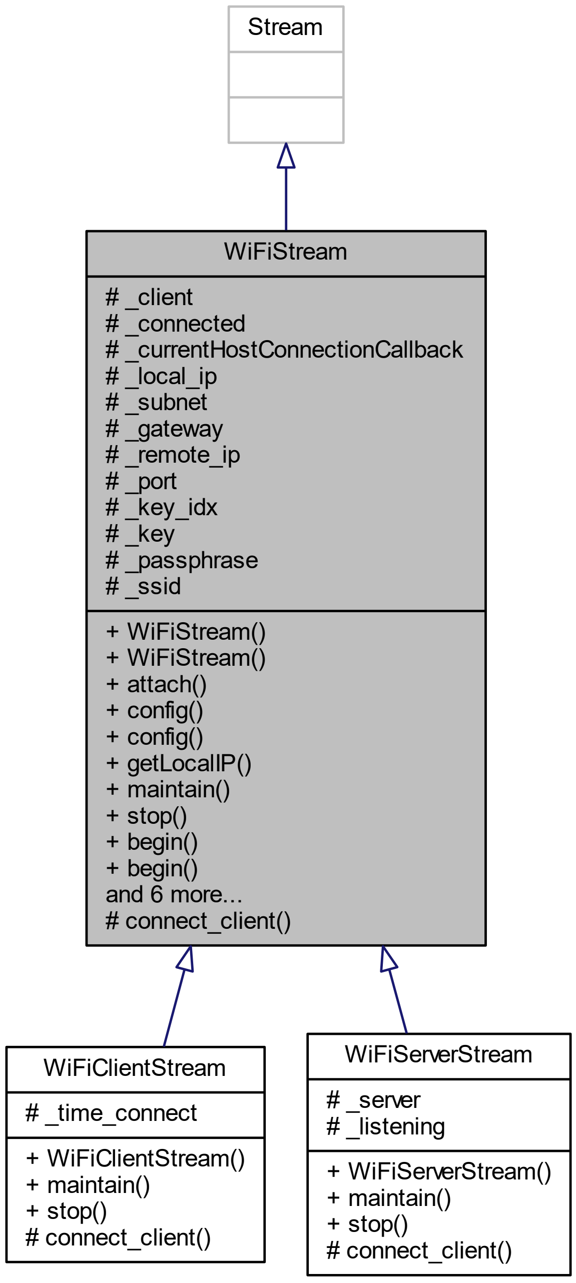
The documentation for this class was generated from the following file:

- WiFiServerStream.h

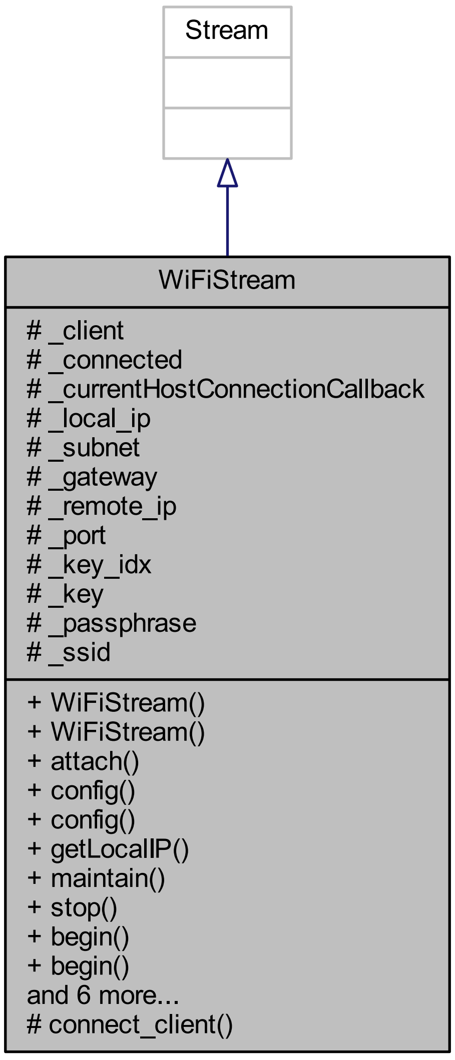
## 26.12 WiFiStream Class Reference

```
#include <WiFiStream.h>
```

Inheritance diagram for WiFiStream:



Collaboration diagram for WiFiStream:



## Public Member Functions

- WiFiStream (uint16_t server_port)
- WiFiStream (IPAddress server_ip, uint16_t server_port)
- void attach (hostConnectionCallbackFunction newFunction)
- void config (IPAddress local_ip)
- void config (IPAddress local_ip, IPAddress gateway, IPAddress subnet)
- IPAddress getLocalIP ()
- virtual bool maintain ()=0
- virtual void stop ()=0
- int begin (char ∗ssid)
- int begin (char ∗ssid, uint8_t key_idx, const char ∗key)
- int begin (char ∗ssid, const char ∗passphrase)

- int available ()
- void flush ()
- int peek ()
- int read ()
- size_t write (uint8_t byte)

## Protected Member Functions

- virtual bool connect_client ()=0

## Protected Attributes

- WiFiClient _client
- bool _connected = false
- hostConnectionCallbackFunction _currentHostConnectionCallback
- IPAddress _local_ip
- IPAddress _subnet
- IPAddress _gateway
- IPAddress _remote_ip
- uint16_t _port
- uint8_t _key_idx
- const char ∗ _key = nullptr
- const char ∗ _passphrase = nullptr
- char ∗ _ssid = nullptr

## 26.12.1 Detailed Description

Definition at line 35 of file WiFiStream.h.

## 26.12.2 Constructor & Destructor Documentation

### 26.12.2.1 WiFiStream() [1/2]

```
WiFiStream::WiFiStream (
            uint16_t server_port )   [inline]
```

constructor for TCP server

Definition at line 61 of file WiFiStream.h.

```
61 : _port(server_port) {}
```

#### 26.12.2.2 WiFiStream() [2/2]

```
WiFiStream::WiFiStream (
             IPAddress server_ip,
             uint16_t server_port )  [inline]
```

constructor for TCP client

Definition at line 64 of file WiFiStream.h.

```
64 : _remote_ip(server_ip), _port(server_port) {}
```

### 26.12.3 Member Function Documentation

#### 26.12.3.1 attach()

```
void WiFiStream::attach (
             hostConnectionCallbackFunction newFunction )  [inline]
```

Definition at line 66 of file WiFiStream.h.

```
66 { _currentHostConnectionCallback = newFunction; }
```

References _currentHostConnectionCallback.

#### 26.12.3.2 available()

```
int WiFiStream::available (
             void  )  [inline]
```

Definition at line 195 of file WiFiStream.h.

```
200   {
```

#### 26.12.3.3 begin() [1/3]

```
int WiFiStream::begin (
             char * ssid )  [inline]
```

initialize WiFi without security (open) and initiate client connection if WiFi connection is already established

**Returns**

WL_CONNECTED if WiFi connection is established

Definition at line 151 of file WiFiStream.h.

```
155   {
156     _ssid = ssid;
157
158     WiFi.begin(ssid);
```

#### 26.12.3.4 begin() [2/3]

```
int WiFiStream::begin (
             char * ssid,
             const char * passphrase )  [inline]
```

initialize WiFi with WPA-PSK security and initiate client connection if WiFi connection is already established

**Returns**

WL_CONNECTED if WiFi connection is established

Definition at line 182 of file WiFiStream.h.

```
{
  _ssid = ssid;
  _passphrase = passphrase;

```

#### 26.12.3.5 begin() [3/3]

```
int WiFiStream::begin (
             char * ssid,
             uint8_t key_idx,
             const char * key )  [inline]
```

initialize WiFi with WEP security and initiate client connection if WiFi connection is already established

**Returns**

WL_CONNECTED if WiFi connection is established

Definition at line 166 of file WiFiStream.h.

```
{
  _ssid = ssid;
  _key_idx = key_idx;
  _key = key;

```

#### 26.12.3.6 config() [1/2]

```
void WiFiStream::config (
             IPAddress local_ip )  [inline]
```

configure a static local IP address without defining the local network DHCP will be used as long as local IP address is not defined

Definition at line 76 of file WiFiStream.h.

```
{
  _local_ip = local_ip;
  WiFi.config( local_ip );
```

#### 26.12.3.7 config() [2/2]

```
void WiFiStream::config (
            IPAddress local_ip,
            IPAddress gateway,
            IPAddress subnet )  [inline]
```

configure a static local IP address DHCP will be used as long as local IP address is not defined

Definition at line 87 of file WiFiStream.h.

```
  {
    _local_ip = local_ip;
    _subnet = subnet;
    _gateway = gateway;
#ifndef ESP8266
    WiFi.config( local_ip, IPAddress(0, 0, 0, 0), gateway, subnet );
#else
    WiFi.config( local_ip, gateway, subnet );
#endif
```

#### 26.12.3.8 connect_client()

```
virtual bool WiFiStream::connect_client ( )  [protected], [pure virtual]
```

check if TCP client is connected

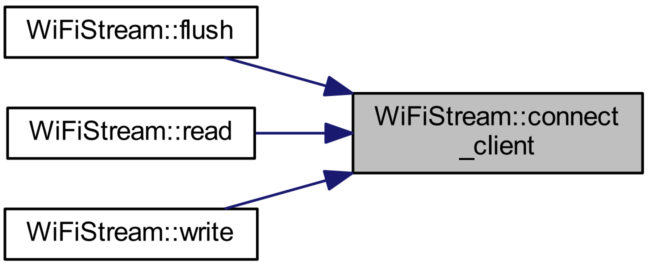
**Returns**

true if connected

Implemented in WiFiClientStream, and WiFiServerStream.

Referenced by flush(), read(), and write().

Here is the caller graph for this function:



#### 26.12.3.9 flush()

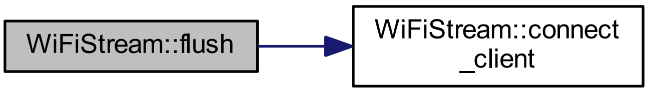
```
void WiFiStream::flush (
            void ) [inline]
```

Definition at line 200 of file WiFiStream.h.

```
{
  return connect_client() ? _client.available() : 0;
}

```

References _client, and connect_client().

Here is the call graph for this function:



#### 26.12.3.10 getLocalIP()

```
IPAddress WiFiStream::getLocalIP ( ) [inline]
```

**Returns**

local IP address

Definition at line 102 of file WiFiStream.h.

```
{
  return WiFi.localIP();
```

#### 26.12.3.11 maintain()

```
virtual bool WiFiStream::maintain ( ) [pure virtual]
```

maintain WiFi and TCP connection

**Returns**

true if WiFi and TCP connection are established

Implemented in WiFiClientStream, and WiFiServerStream.

### 26.12.3.12 peek()

```
int WiFiStream::peek (
             void  )  [inline]
```

Definition at line 205 of file WiFiStream.h.

```
{
  if( _client ) _client.flush();
}
```

References _client.

### 26.12.3.13 read()

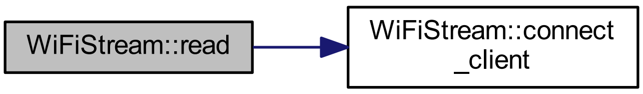
```
int WiFiStream::read (
             void  )  [inline]
```

Definition at line 210 of file WiFiStream.h.

```
{
  return connect_client() ? _client.peek(): 0;
}
```

References _client, and connect_client().

Here is the call graph for this function:



### 26.12.3.14 stop()

```
virtual void WiFiStream::stop ( )  [pure virtual]
```

close TCP client connection

Implemented in WiFiServerStream, and WiFiClientStream.

#### 26.12.3.15 write()

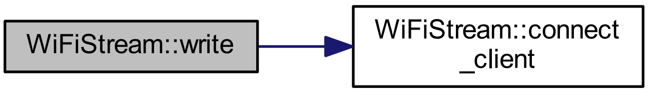
```
size_t WiFiStream::write (
            uint8_t byte )  [inline]
```

Definition at line 215 of file WiFiStream.h.

```
{
  return connect_client() ? _client.read() : -1;
}

```

References _client, and connect_client().

Here is the call graph for this function:



### 26.12.4 Field Documentation

#### 26.12.4.1 _client

```
WiFiClient WiFiStream::_client  [protected]
```

Definition at line 38 of file WiFiStream.h.

Referenced by WiFiServerStream::connect_client(), WiFiClientStream::connect_client(), flush(), peek(), read(), WiFiClientStream::stop(), WiFiServerStream::stop(), and write().

#### 26.12.4.2 _connected

```
bool WiFiStream::_connected = false  [protected]
```

Definition at line 39 of file WiFiStream.h.

Referenced by WiFiServerStream::connect_client(), WiFiClientStream::connect_client(), WiFiClientStream::stop(), and WiFiServerStream::stop().

#### 26.12.4.3 _currentHostConnectionCallback

```
hostConnectionCallbackFunction WiFiStream::_currentHostConnectionCallback  [protected]
```

Definition at line 40 of file WiFiStream.h.

Referenced by attach(), WiFiServerStream::connect_client(), WiFiClientStream::connect_client(), WiFiClient↩ Stream::stop(), and WiFiServerStream::stop().

#### 26.12.4.4 _gateway

```
IPAddress WiFiStream::_gateway  [protected]
```

Definition at line 45 of file WiFiStream.h.

#### 26.12.4.5 _key

```
const char* WiFiStream::_key = nullptr  [protected]
```

Definition at line 49 of file WiFiStream.h.

#### 26.12.4.6 _key_idx

```
uint8_t WiFiStream::_key_idx  [protected]
```

Definition at line 48 of file WiFiStream.h.

#### 26.12.4.7 _local_ip

```
IPAddress WiFiStream::_local_ip  [protected]
```

Definition at line 43 of file WiFiStream.h.

#### 26.12.4.8 _passphrase

```
const char* WiFiStream::_passphrase = nullptr  [protected]
```

Definition at line 50 of file WiFiStream.h.

#### 26.12.4.9 _port

```
uint16_t WiFiStream::_port  [protected]
```

Definition at line 47 of file WiFiStream.h.

Referenced by WiFiClientStream::connect_client(), and WiFiServerStream::maintain().

#### 26.12.4.10 _remote_ip

```
IPAddress WiFiStream::_remote_ip  [protected]
```

Definition at line 46 of file WiFiStream.h.

Referenced by WiFiClientStream::connect_client().

#### 26.12.4.11 _ssid

```
char* WiFiStream::_ssid = nullptr  [protected]
```

Definition at line 51 of file WiFiStream.h.

#### 26.12.4.12 _subnet

```
IPAddress WiFiStream::_subnet  [protected]
```

Definition at line 44 of file WiFiStream.h.

The documentation for this class was generated from the following file:

- WiFiStream.h

# Chapter 27

# File Documentation

## 27.1 accelStepperFirmata.md File Reference

## 27.2 bleConfig.h File Reference

### Macros

- #define FIRMATA_BLE_LOCAL_NAME "FIRMATA"
- #define FIRMATA_BLE_MIN_INTERVAL 0x0006
- #define FIRMATA_BLE_MAX_INTERVAL 0x0018
- #define FIRMATA_BLE_TXBUFFER_FLUSH_INTERVAL 30

### 27.2.1 Macro Definition Documentation

#### 27.2.1.1 FIRMATA_BLE_LOCAL_NAME

```
#define FIRMATA_BLE_LOCAL_NAME "FIRMATA"
```

Definition at line 21 of file bleConfig.h.

#### 27.2.1.2 FIRMATA_BLE_MAX_INTERVAL

```
#define FIRMATA_BLE_MAX_INTERVAL 0x0018
```

Definition at line 98 of file bleConfig.h.

#### 27.2.1.3 FIRMATA_BLE_MIN_INTERVAL

```
#define FIRMATA_BLE_MIN_INTERVAL 0x0006
```

Definition at line 97 of file bleConfig.h.

#### 27.2.1.4 FIRMATA_BLE_TXBUFFER_FLUSH_INTERVAL

```
#define FIRMATA_BLE_TXBUFFER_FLUSH_INTERVAL 30
```

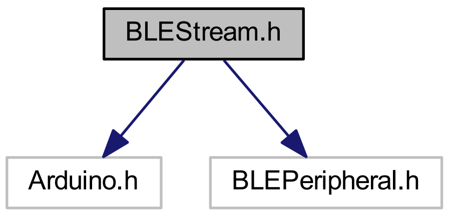
Definition at line 102 of file bleConfig.h.

## 27.3 BLEStream.cpp File Reference

## 27.4 BLEStream.h File Reference

```
#include <Arduino.h>
#include <BLEPeripheral.h>
```

Include dependency graph for BLEStream.h:



### Data Structures

- class BLEStream

### Macros

- #define _MAX_ATTR_DATA_LEN_ BLE_ATTRIBUTE_MAX_VALUE_LENGTH
- #define BLESTREAM_TXBUFFER_FLUSH_INTERVAL 80
- #define BLESTREAM_MIN_FLUSH_INTERVAL 8

### 27.4.1 Macro Definition Documentation

#### 27.4.1.1 _MAX_ATTR_DATA_LEN_

```
#define _MAX_ATTR_DATA_LEN_ BLE_ATTRIBUTE_MAX_VALUE_LENGTH
```

Definition at line 19 of file BLEStream.h.

#### 27.4.1.2 BLESTREAM_MIN_FLUSH_INTERVAL

```
#define BLESTREAM_MIN_FLUSH_INTERVAL 8
```

Definition at line 23 of file BLEStream.h.

#### 27.4.1.3 BLESTREAM_TXBUFFER_FLUSH_INTERVAL

```
#define BLESTREAM_TXBUFFER_FLUSH_INTERVAL 80
```

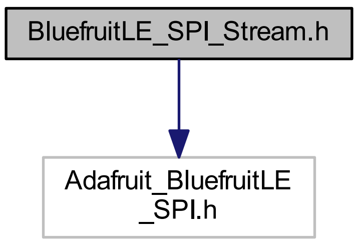
Definition at line 22 of file BLEStream.h.

## 27.5 BluefruitLE_SPI_Stream.cpp File Reference

## 27.6 BluefruitLE_SPI_Stream.h File Reference

```
#include <Adafruit_BluefruitLE_SPI.h>
```

Include dependency graph for BluefruitLE_SPI_Stream.h:



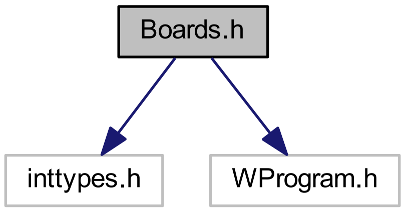
### Data Structures

- class BluefruitLE_SPI_Stream

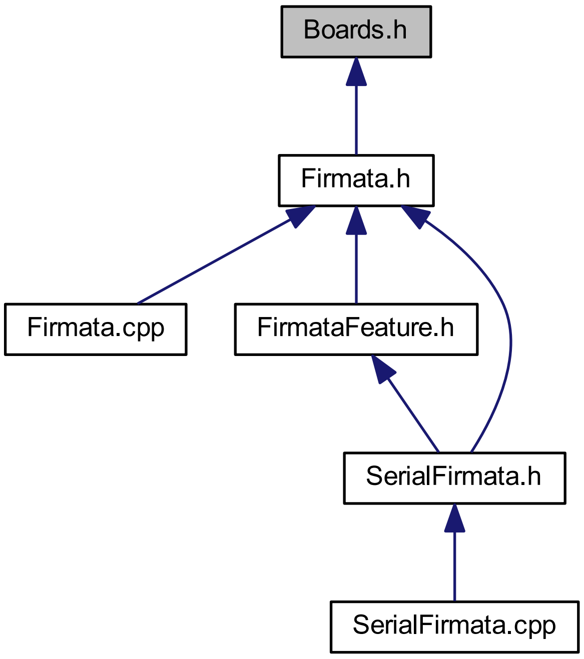
## 27.7 Boards.h File Reference

```
#include <inttypes.h>
#include "WProgram.h"
```

Include dependency graph for Boards.h:



This graph shows which files directly or indirectly include this file:



## Macros

- #define MAX_SERVOS 0
- #define digitalPinHasPWM(p) IS_PIN_DIGITAL(p)
- #define IS_PIN_SPI(p) 0
- #define IS_PIN_SERIAL(p) 0
- #define DEFAULT_PWM_RESOLUTION 8
- #define TOTAL_PORTS ((TOTAL_PINS + 7) / 8)

## Functions

- static unsigned char readPort (byte, byte) __attribute__((always_inline
- static unsigned char writePort (byte, byte, byte) __attribute__((always_inline

## Variables

- static unsigned char unused

## 27.7.1 Macro Definition Documentation

### 27.7.1.1 DEFAULT_PWM_RESOLUTION

```
#define DEFAULT_PWM_RESOLUTION 8
```

Definition at line 933 of file Boards.h.

### 27.7.1.2 digitalPinHasPWM

```
#define digitalPinHasPWM(
              p ) IS_PIN_DIGITAL(p)
```

Definition at line 141 of file Boards.h.

### 27.7.1.3 IS_PIN_SERIAL

```
#define IS_PIN_SERIAL(
              p ) 0
```

Definition at line 929 of file Boards.h.

#### 27.7.1.4 IS_PIN_SPI

```
#define IS_PIN_SPI(
             p ) 0
```

Definition at line 925 of file Boards.h.

#### 27.7.1.5 MAX_SERVOS

```
#define MAX_SERVOS 0
```

Definition at line 32 of file Boards.h.

#### 27.7.1.6 TOTAL_PORTS

```
#define TOTAL_PORTS ((TOTAL_PINS + 7) / 8)
```

Definition at line 1013 of file Boards.h.

## 27.7.2 Function Documentation

#### 27.7.2.1 readPort()

```
static unsigned char readPort (
            byte port,
            byte bitmask )  [inline], [static]
```

Definition at line 941 of file Boards.h.

```
{
#if defined(ARDUINO_PINOUT_OPTIMIZE)
  if (port == 0) return (PIND & 0xFC) & bitmask; // ignore Rx/Tx 0/1
  if (port == 1) return ((PINB & 0x3F) | ((PINC & 0x03) « 6)) & bitmask;
  if (port == 2) return ((PINC & 0x3C) » 2) & bitmask;
  return 0;
#else
  unsigned char out = 0, pin = port * 8;
  if (IS_PIN_DIGITAL(pin + 0) && (bitmask & 0x01) && digitalRead(PIN_TO_DIGITAL(pin + 0))) out |= 0x01;
  if (IS_PIN_DIGITAL(pin + 1) && (bitmask & 0x02) && digitalRead(PIN_TO_DIGITAL(pin + 1))) out |= 0x02;
  if (IS_PIN_DIGITAL(pin + 2) && (bitmask & 0x04) && digitalRead(PIN_TO_DIGITAL(pin + 2))) out |= 0x04;
  if (IS_PIN_DIGITAL(pin + 3) && (bitmask & 0x08) && digitalRead(PIN_TO_DIGITAL(pin + 3))) out |= 0x08;
  if (IS_PIN_DIGITAL(pin + 4) && (bitmask & 0x10) && digitalRead(PIN_TO_DIGITAL(pin + 4))) out |= 0x10;
  if (IS_PIN_DIGITAL(pin + 5) && (bitmask & 0x20) && digitalRead(PIN_TO_DIGITAL(pin + 5))) out |= 0x20;
  if (IS_PIN_DIGITAL(pin + 6) && (bitmask & 0x40) && digitalRead(PIN_TO_DIGITAL(pin + 6))) out |= 0x40;
  if (IS_PIN_DIGITAL(pin + 7) && (bitmask & 0x80) && digitalRead(PIN_TO_DIGITAL(pin + 7))) out |= 0x80;
  return out;
#endif
}
```

#### 27.7.2.2 writePort()

```
static unsigned char writePort (
            byte port,
            byte value,
            byte bitmask )  [inline], [static]
```

Definition at line 967 of file Boards.h.

```
{
#if defined(ARDUINO_PINOUT_OPTIMIZE)
  if (port == 0) {
    bitmask = bitmask & 0xFC;  // do not touch Tx & Rx pins
    byte valD = value & bitmask;
    byte maskD =  bitmask;
    cli();
    PORTD = (PORTD & maskD) | valD;
    sei();
  } else if (port == 1) {
    byte valB = (value & bitmask) & 0x3F;
    byte valC = (value & bitmask) » 6;
    byte maskB =  (bitmask & 0x3F);
    byte maskC =  ((bitmask & 0xC0) » 6);
    cli();
    PORTB = (PORTB & maskB) | valB;
    PORTC = (PORTC & maskC) | valC;
    sei();
  } else if (port == 2) {
    bitmask = bitmask & 0x0F;
    byte valC = (value & bitmask) « 2;
    byte maskC =  (bitmask « 2);
    cli();
    PORTC = (PORTC & maskC) | valC;
    sei();
  }
  return 1;
#else
  byte pin = port * 8;
  if ((bitmask & 0x01)) digitalWrite(PIN_TO_DIGITAL(pin + 0), (value & 0x01));
  if ((bitmask & 0x02)) digitalWrite(PIN_TO_DIGITAL(pin + 1), (value & 0x02));
  if ((bitmask & 0x04)) digitalWrite(PIN_TO_DIGITAL(pin + 2), (value & 0x04));
  if ((bitmask & 0x08)) digitalWrite(PIN_TO_DIGITAL(pin + 3), (value & 0x08));
  if ((bitmask & 0x10)) digitalWrite(PIN_TO_DIGITAL(pin + 4), (value & 0x10));
  if ((bitmask & 0x20)) digitalWrite(PIN_TO_DIGITAL(pin + 5), (value & 0x20));
  if ((bitmask & 0x40)) digitalWrite(PIN_TO_DIGITAL(pin + 6), (value & 0x40));
  if ((bitmask & 0x80)) digitalWrite(PIN_TO_DIGITAL(pin + 7), (value & 0x80));
  return 1;
#endif
}
```

### 27.7.3 Variable Documentation

#### 27.7.3.1 unused

```
static unsigned char unused
```

Definition at line 940 of file Boards.h.

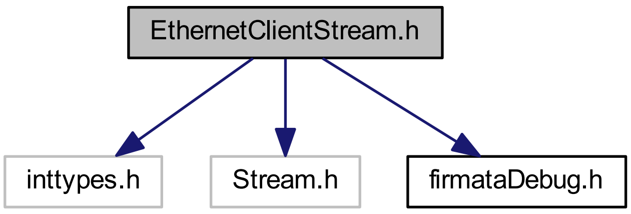
## 27.8 encoder.md File Reference

## 27.9 EthernetClientStream.cpp File Reference

## 27.10 EthernetClientStream.h File Reference

```
#include <inttypes.h>
#include <Stream.h>
#include "firmataDebug.h"
```

Include dependency graph for EthernetClientStream.h:



### Data Structures

- class EthernetClientStream

### Macros

- #define MILLIS_RECONNECT 5000

### 27.10.1 Macro Definition Documentation

#### 27.10.1.1 MILLIS_RECONNECT

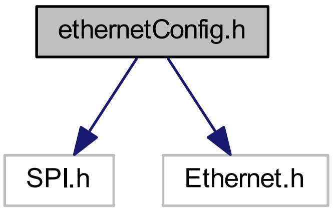
```
#define MILLIS_RECONNECT 5000
```

Definition at line 29 of file EthernetClientStream.h.

## 27.11 ethernetConfig.h File Reference

```
#include <SPI.h>
#include <Ethernet.h>
```

Include dependency graph for ethernetConfig.h:



### Macros

- #define WIZ5100_ETHERNET
- #define remote_ip IPAddress(10, 0, 0, 3)
- #define network_port 3030
- #define local_ip IPAddress(10, 0, 0, 15)
- #define IS_IGNORE_PIN(p) ((IS_PIN_SPI(p) || (p) == 4) || (p) == 10)

### Variables

- EthernetClient client
- const byte mac [ ] = {0x90, 0xA2, 0xDA, 0x00, 0x53, 0xE5}

### 27.11.1 Macro Definition Documentation

#### 27.11.1.1 IS_IGNORE_PIN

```
#define IS_IGNORE_PIN(
              p ) ((IS_PIN_SPI(p) || (p) == 4) || (p) == 10)
```

Definition at line 92 of file ethernetConfig.h.

#### 27.11.1.2 local_ip

```
#define local_ip IPAddress(10, 0, 0, 15)
```

Definition at line 67 of file ethernetConfig.h.

#### 27.11.1.3 network_port

```
#define network_port 3030
```

Definition at line 62 of file ethernetConfig.h.

#### 27.11.1.4 remote_ip

```
#define remote_ip IPAddress(10, 0, 0, 3)
```

Definition at line 55 of file ethernetConfig.h.

#### 27.11.1.5 WIZ5100_ETHERNET

```
#define WIZ5100_ETHERNET
```

Definition at line 20 of file ethernetConfig.h.

### 27.11.2 Variable Documentation

#### 27.11.2.1 client

```
EthernetClient client
```

Definition at line 25 of file ethernetConfig.h.

#### 27.11.2.2 mac

```
const byte mac[] = {0x90, 0xA2, 0xDA, 0x00, 0x53, 0xE5}
```

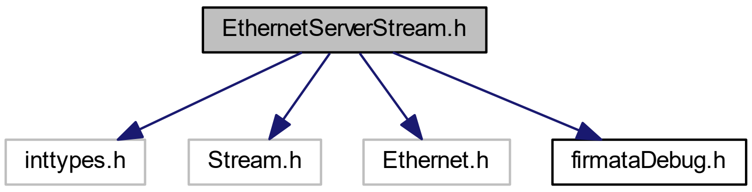
Definition at line 71 of file ethernetConfig.h.

## 27.12 EthernetServerStream.cpp File Reference

## 27.13 EthernetServerStream.h File Reference

```
#include <inttypes.h>
#include <Stream.h>
#include <Ethernet.h>
#include "firmataDebug.h"
```

Include dependency graph for EthernetServerStream.h:



### Data Structures

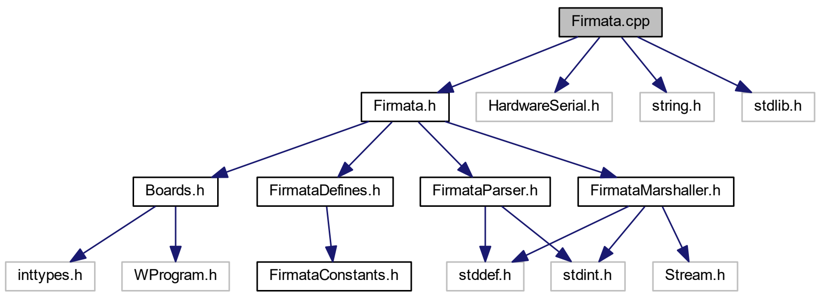
- class EthernetServerStream

## 27.14 feature-registry.md File Reference

## 27.15 Firmata.cpp File Reference

```
#include "Firmata.h"
#include "HardwareSerial.h"
#include <string.h>
#include <stdlib.h>
```

Include dependency graph for Firmata.cpp:



## Variables

- FirmataClass Firmata

### 27.15.1 Variable Documentation

#### 27.15.1.1 Firmata

```
FirmataClass Firmata
```

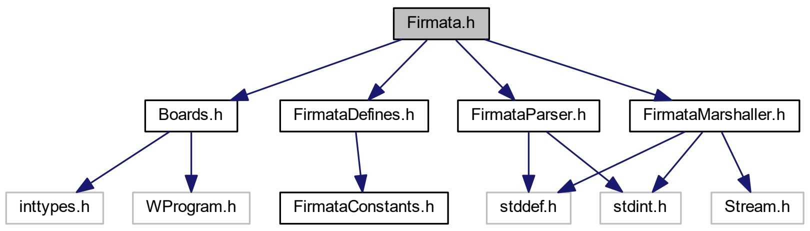
Definition at line 30 of file Firmata.cpp.

Referenced by SerialFirmata::checkSerial(), SerialFirmata::handleCapability(), SerialFirmata::handlePinMode(), and SerialFirmata::handleSysex().

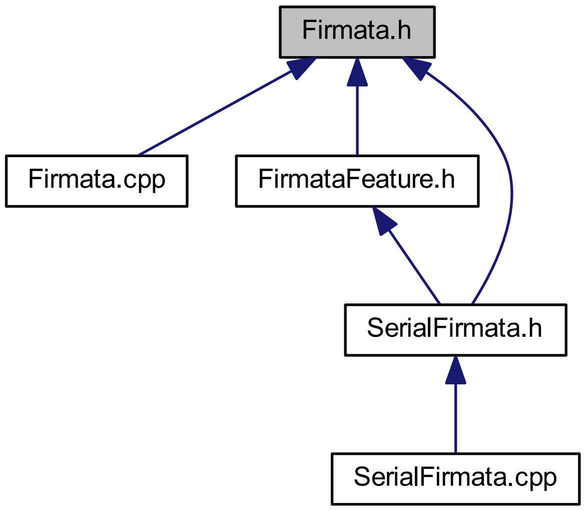
## 27.16 Firmata.h File Reference

```
#include "Boards.h"
#include "FirmataDefines.h"
#include "FirmataMarshaller.h"
#include "FirmataParser.h"
```

Include dependency graph for Firmata.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class firmata::FirmataClass

## Namespaces

- firmata

## Macros

- #define FIRMATA_MAJOR_VERSION 2
- #define FIRMATA_MINOR_VERSION 5
- #define FIRMATA_BUGFIX_VERSION 1
- #define FIRMATA_STRING 0x71
- #define SYSEX_I2C_REQUEST 0x76
- #define SYSEX_I2C_REPLY 0x77
- #define SYSEX_SAMPLING_INTERVAL 0x7A
- #define ANALOG 0x02
- #define PWM 0x03
- #define SERVO 0x04
- #define SHIFT 0x05
- #define I2C 0x06
- #define ONEWIRE 0x07
- #define STEPPER 0x08
- #define ENCODER 0x09
- #define IGNORE 0x7F
- #define setFirmwareVersion(x, y) setFirmwareNameAndVersion(__FILE__, x, y)

## Typedefs

- typedef firmata::FirmataClass::callbackFunction callbackFunction
- typedef firmata::FirmataClass::systemCallbackFunction systemCallbackFunction
- typedef firmata::FirmataClass::stringCallbackFunction stringCallbackFunction
- typedef firmata::FirmataClass::sysexCallbackFunction sysexCallbackFunction

## Variables

- firmata::FirmataClass Firmata

## 27.16.1 Macro Definition Documentation

### 27.16.1.1 ANALOG

```
#define ANALOG 0x02
```

Definition at line 41 of file Firmata.h.

### 27.16.1.2 ENCODER

```
#define ENCODER 0x09
```

Definition at line 48 of file Firmata.h.

### 27.16.1.3 FIRMATA_BUGFIX_VERSION

```
#define FIRMATA_BUGFIX_VERSION 1
```

Definition at line 27 of file Firmata.h.

### 27.16.1.4 FIRMATA_MAJOR_VERSION

```
#define FIRMATA_MAJOR_VERSION 2
```

Definition at line 25 of file Firmata.h.

#### 27.16.1.5 FIRMATA_MINOR_VERSION

```
#define FIRMATA_MINOR_VERSION 5
```

Definition at line 26 of file Firmata.h.

#### 27.16.1.6 FIRMATA_STRING

```
#define FIRMATA_STRING 0x71
```

Definition at line 32 of file Firmata.h.

#### 27.16.1.7 I2C

```
#define I2C 0x06
```

Definition at line 45 of file Firmata.h.

#### 27.16.1.8 IGNORE

```
#define IGNORE 0x7F
```

Definition at line 49 of file Firmata.h.

#### 27.16.1.9 ONEWIRE

```
#define ONEWIRE 0x07
```

Definition at line 46 of file Firmata.h.

#### 27.16.1.10 PWM

```
#define PWM 0x03
```

Definition at line 42 of file Firmata.h.

#### 27.16.1.11 SERVO

```
#define SERVO 0x04
```

Definition at line 43 of file Firmata.h.

#### 27.16.1.12 setFirmwareVersion

```
#define setFirmwareVersion(
             x,
             y ) setFirmwareNameAndVersion(__FILE__, x, y)
```

Definition at line 178 of file Firmata.h.

#### 27.16.1.13 SHIFT

```
#define SHIFT 0x05
```

Definition at line 44 of file Firmata.h.

#### 27.16.1.14 STEPPER

```
#define STEPPER 0x08
```

Definition at line 47 of file Firmata.h.

#### 27.16.1.15 SYSEX_I2C_REPLY

```
#define SYSEX_I2C_REPLY 0x77
```

Definition at line 34 of file Firmata.h.

#### 27.16.1.16 SYSEX_I2C_REQUEST

```
#define SYSEX_I2C_REQUEST 0x76
```

Definition at line 33 of file Firmata.h.

#### 27.16.1.17 SYSEX_SAMPLING_INTERVAL

```
#define SYSEX_SAMPLING_INTERVAL 0x7A
```

Definition at line 35 of file Firmata.h.

## 27.16.2 Typedef Documentation

#### 27.16.2.1 callbackFunction

```
typedef firmata::FirmataClass::callbackFunction callbackFunction
```

Definition at line 162 of file Firmata.h.

#### 27.16.2.2 stringCallbackFunction

```
typedef firmata::FirmataClass::stringCallbackFunction stringCallbackFunction
```

Definition at line 164 of file Firmata.h.

#### 27.16.2.3 sysexCallbackFunction

```
typedef firmata::FirmataClass::sysexCallbackFunction sysexCallbackFunction
```

Definition at line 165 of file Firmata.h.

#### 27.16.2.4 systemCallbackFunction

```
typedef firmata::FirmataClass::systemCallbackFunction systemCallbackFunction
```

Definition at line 163 of file Firmata.h.

## 27.16.3 Variable Documentation

#### 27.16.3.1 Firmata

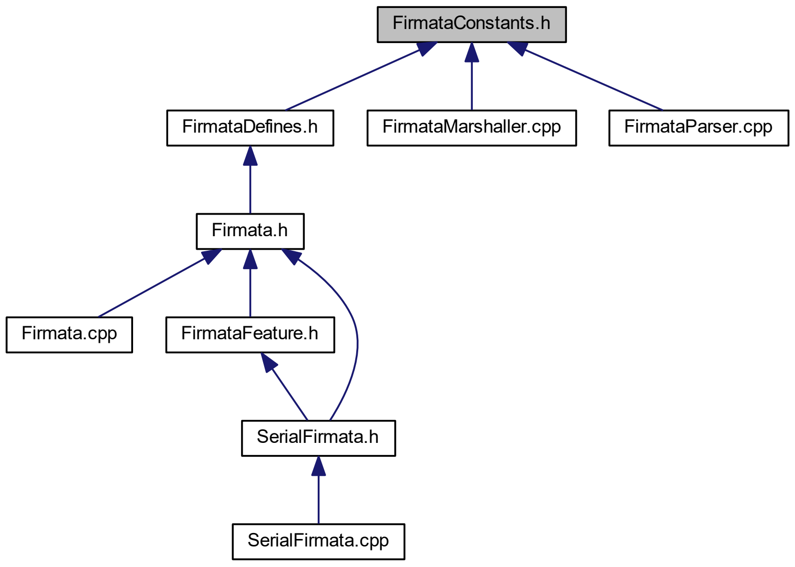
```
firmata::FirmataClass Firmata
```

Definition at line 30 of file Firmata.cpp.

Referenced by SerialFirmata::checkSerial(), SerialFirmata::handleCapability(), SerialFirmata::handlePinMode(), and SerialFirmata::handleSysex().

## 27.17 FirmataConstants.h File Reference

This graph shows which files directly or indirectly include this file:



### Namespaces

- firmata

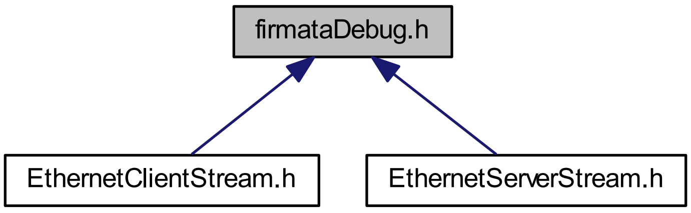
### Variables

- static const int firmata::FIRMWARE_MAJOR_VERSION = 2
- static const int firmata::FIRMWARE_MINOR_VERSION = 5
- static const int firmata::FIRMWARE_BUGFIX_VERSION = 7
- static const int firmata::PROTOCOL_MAJOR_VERSION = 2
- static const int firmata::PROTOCOL_MINOR_VERSION = 5
- static const int firmata::PROTOCOL_BUGFIX_VERSION = 1
- static const int firmata::MAX_DATA_BYTES = 64
- static const int firmata::DIGITAL_MESSAGE = 0x90
- static const int firmata::ANALOG_MESSAGE = 0xE0

- static const int firmata::REPORT_ANALOG = 0xC0
- static const int firmata::REPORT_DIGITAL = 0xD0
- static const int firmata::SET_PIN_MODE = 0xF4
- static const int firmata::SET_DIGITAL_PIN_VALUE = 0xF5
- static const int firmata::REPORT_VERSION = 0xF9
- static const int firmata::SYSTEM_RESET = 0xFF
- static const int firmata::START_SYSEX = 0xF0
- static const int firmata::END_SYSEX = 0xF7
- static const int firmata::SERIAL_DATA = 0x60
- static const int firmata::ENCODER_DATA = 0x61
- static const int firmata::SERVO_CONFIG = 0x70
- static const int firmata::STRING_DATA = 0x71
- static const int firmata::STEPPER_DATA = 0x72
- static const int firmata::ONEWIRE_DATA = 0x73
- static const int firmata::SHIFT_DATA = 0x75
- static const int firmata::I2C_REQUEST = 0x76
- static const int firmata::I2C_REPLY = 0x77
- static const int firmata::I2C_CONFIG = 0x78
- static const int firmata::REPORT_FIRMWARE = 0x79
- static const int firmata::EXTENDED_ANALOG = 0x6F
- static const int firmata::PIN_STATE_QUERY = 0x6D
- static const int firmata::PIN_STATE_RESPONSE = 0x6E
- static const int firmata::CAPABILITY_QUERY = 0x6B
- static const int firmata::CAPABILITY_RESPONSE = 0x6C
- static const int firmata::ANALOG_MAPPING_QUERY = 0x69
- static const int firmata::ANALOG_MAPPING_RESPONSE = 0x6A
- static const int firmata::SAMPLING_INTERVAL = 0x7A
- static const int firmata::SCHEDULER_DATA = 0x7B
- static const int firmata::SYSEX_NON_REALTIME = 0x7E
- static const int firmata::SYSEX_REALTIME = 0x7F
- static const int firmata::PIN_MODE_INPUT = 0x00
- static const int firmata::PIN_MODE_OUTPUT = 0x01
- static const int firmata::PIN_MODE_ANALOG = 0x02
- static const int firmata::PIN_MODE_PWM = 0x03
- static const int firmata::PIN_MODE_SERVO = 0x04
- static const int firmata::PIN_MODE_SHIFT = 0x05
- static const int firmata::PIN_MODE_I2C = 0x06
- static const int firmata::PIN_MODE_ONEWIRE = 0x07
- static const int firmata::PIN_MODE_STEPPER = 0x08
- static const int firmata::PIN_MODE_ENCODER = 0x09
- static const int firmata::PIN_MODE_SERIAL = 0x0A
- static const int firmata::PIN_MODE_PULLUP = 0x0B
- static const int firmata::PIN_MODE_IGNORE = 0x7F
- static const int firmata::TOTAL_PIN_MODES = 13

## 27.18 firmataDebug.h File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- #define DEBUG_BEGIN(baud)
- #define DEBUG_PRINTLN(x)
- #define DEBUG_PRINT(x)

### 27.18.1 Macro Definition Documentation

#### 27.18.1.1 DEBUG_BEGIN

```
#define DEBUG_BEGIN(
              baud )
```

Definition at line 9 of file firmataDebug.h.

#### 27.18.1.2 DEBUG_PRINT

```
#define DEBUG_PRINT(
              x )
```

Definition at line 11 of file firmataDebug.h.

#### 27.18.1.3 DEBUG_PRINTLN

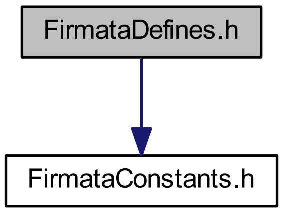
```
#define DEBUG_PRINTLN(
              x )
```

Definition at line 10 of file firmataDebug.h.

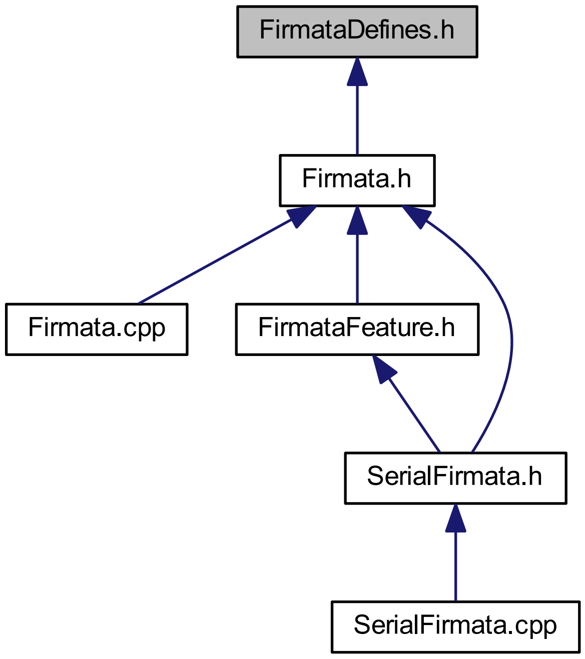
## 27.19 FirmataDefines.h File Reference

```
#include "FirmataConstants.h"
```

Include dependency graph for FirmataDefines.h:



This graph shows which files directly or indirectly include this file:



## Macros

- #define FIRMATA_FIRMWARE_MAJOR_VERSION firmata::FIRMWARE_MAJOR_VERSION
- #define FIRMATA_FIRMWARE_MINOR_VERSION firmata::FIRMWARE_MINOR_VERSION
- #define FIRMATA_FIRMWARE_BUGFIX_VERSION firmata::FIRMWARE_BUGFIX_VERSION
- #define FIRMATA_PROTOCOL_MAJOR_VERSION firmata::PROTOCOL_MAJOR_VERSION
- #define FIRMATA_PROTOCOL_MINOR_VERSION firmata::PROTOCOL_MINOR_VERSION
- #define FIRMATA_PROTOCOL_BUGFIX_VERSION firmata::PROTOCOL_BUGFIX_VERSION
- #define MAX_DATA_BYTES firmata::MAX_DATA_BYTES
- #define DIGITAL_MESSAGE firmata::DIGITAL_MESSAGE
- #define ANALOG_MESSAGE firmata::ANALOG_MESSAGE
- #define REPORT_ANALOG firmata::REPORT_ANALOG
- #define REPORT_DIGITAL firmata::REPORT_DIGITAL
- #define SET_PIN_MODE firmata::SET_PIN_MODE
- #define SET_DIGITAL_PIN_VALUE firmata::SET_DIGITAL_PIN_VALUE
- #define REPORT_VERSION firmata::REPORT_VERSION
- #define SYSTEM_RESET firmata::SYSTEM_RESET
- #define START_SYSEX firmata::START_SYSEX
- #define END_SYSEX firmata::END_SYSEX
- #define SERIAL_MESSAGE firmata::SERIAL_DATA
- #define ENCODER_DATA firmata::ENCODER_DATA
- #define SERVO_CONFIG firmata::SERVO_CONFIG
- #define STRING_DATA firmata::STRING_DATA
- #define STEPPER_DATA firmata::STEPPER_DATA
- #define ONEWIRE_DATA firmata::ONEWIRE_DATA
- #define SHIFT_DATA firmata::SHIFT_DATA
- #define I2C_REQUEST firmata::I2C_REQUEST
- #define I2C_REPLY firmata::I2C_REPLY
- #define I2C_CONFIG firmata::I2C_CONFIG
- #define REPORT_FIRMWARE firmata::REPORT_FIRMWARE
- #define EXTENDED_ANALOG firmata::EXTENDED_ANALOG
- #define PIN_STATE_QUERY firmata::PIN_STATE_QUERY
- #define PIN_STATE_RESPONSE firmata::PIN_STATE_RESPONSE
- #define CAPABILITY_QUERY firmata::CAPABILITY_QUERY
- #define CAPABILITY_RESPONSE firmata::CAPABILITY_RESPONSE
- #define ANALOG_MAPPING_QUERY firmata::ANALOG_MAPPING_QUERY
- #define ANALOG_MAPPING_RESPONSE firmata::ANALOG_MAPPING_RESPONSE
- #define SAMPLING_INTERVAL firmata::SAMPLING_INTERVAL
- #define SCHEDULER_DATA firmata::SCHEDULER_DATA
- #define SYSEX_NON_REALTIME firmata::SYSEX_NON_REALTIME
- #define SYSEX_REALTIME firmata::SYSEX_REALTIME
- #define PIN_MODE_INPUT firmata::PIN_MODE_INPUT
- #define PIN_MODE_OUTPUT firmata::PIN_MODE_OUTPUT
- #define PIN_MODE_ANALOG firmata::PIN_MODE_ANALOG
- #define PIN_MODE_PWM firmata::PIN_MODE_PWM
- #define PIN_MODE_SERVO firmata::PIN_MODE_SERVO
- #define PIN_MODE_SHIFT firmata::PIN_MODE_SHIFT
- #define PIN_MODE_I2C firmata::PIN_MODE_I2C
- #define PIN_MODE_ONEWIRE firmata::PIN_MODE_ONEWIRE
- #define PIN_MODE_STEPPER firmata::PIN_MODE_STEPPER
- #define PIN_MODE_ENCODER firmata::PIN_MODE_ENCODER
- #define PIN_MODE_SERIAL firmata::PIN_MODE_SERIAL
- #define PIN_MODE_PULLUP firmata::PIN_MODE_PULLUP
- #define PIN_MODE_IGNORE firmata::PIN_MODE_IGNORE
- #define TOTAL_PIN_MODES firmata::TOTAL_PIN_MODES

### 27.19.1 Macro Definition Documentation

#### 27.19.1.1 ANALOG_MAPPING_QUERY

```
#define ANALOG_MAPPING_QUERY firmata::ANALOG_MAPPING_QUERY
```

Definition at line 184 of file FirmataDefines.h.

#### 27.19.1.2 ANALOG_MAPPING_RESPONSE

```
#define ANALOG_MAPPING_RESPONSE firmata::ANALOG_MAPPING_RESPONSE
```

Definition at line 189 of file FirmataDefines.h.

#### 27.19.1.3 ANALOG_MESSAGE

```
#define ANALOG_MESSAGE firmata::ANALOG_MESSAGE
```

Definition at line 50 of file FirmataDefines.h.

#### 27.19.1.4 CAPABILITY_QUERY

```
#define CAPABILITY_QUERY firmata::CAPABILITY_QUERY
```

Definition at line 174 of file FirmataDefines.h.

#### 27.19.1.5 CAPABILITY_RESPONSE

```
#define CAPABILITY_RESPONSE firmata::CAPABILITY_RESPONSE
```

Definition at line 179 of file FirmataDefines.h.

#### 27.19.1.6 DIGITAL_MESSAGE

```
#define DIGITAL_MESSAGE firmata::DIGITAL_MESSAGE
```

Definition at line 45 of file FirmataDefines.h.

#### 27.19.1.7 ENCODER_DATA

```
#define ENCODER_DATA firmata::ENCODER_DATA
```

Definition at line 109 of file FirmataDefines.h.

#### 27.19.1.8 END_SYSEX

```
#define END_SYSEX firmata::END_SYSEX
```

Definition at line 96 of file FirmataDefines.h.

#### 27.19.1.9 EXTENDED_ANALOG

```
#define EXTENDED_ANALOG firmata::EXTENDED_ANALOG
```

Definition at line 159 of file FirmataDefines.h.

#### 27.19.1.10 FIRMATA_FIRMWARE_BUGFIX_VERSION

```
#define FIRMATA_FIRMWARE_BUGFIX_VERSION firmata::FIRMWARE_BUGFIX_VERSION
```

Definition at line 25 of file FirmataDefines.h.

#### 27.19.1.11 FIRMATA_FIRMWARE_MAJOR_VERSION

```
#define FIRMATA_FIRMWARE_MAJOR_VERSION firmata::FIRMWARE_MAJOR_VERSION
```

Definition at line 23 of file FirmataDefines.h.

#### 27.19.1.12 FIRMATA_FIRMWARE_MINOR_VERSION

```
#define FIRMATA_FIRMWARE_MINOR_VERSION firmata::FIRMWARE_MINOR_VERSION
```

Definition at line 24 of file FirmataDefines.h.

#### 27.19.1.13 FIRMATA_PROTOCOL_BUGFIX_VERSION

```
#define FIRMATA_PROTOCOL_BUGFIX_VERSION firmata::PROTOCOL_BUGFIX_VERSION
```

Definition at line 33 of file FirmataDefines.h.

#### 27.19.1.14 FIRMATA_PROTOCOL_MAJOR_VERSION

```
#define FIRMATA_PROTOCOL_MAJOR_VERSION firmata::PROTOCOL_MAJOR_VERSION
```

Definition at line 31 of file FirmataDefines.h.

#### 27.19.1.15 FIRMATA_PROTOCOL_MINOR_VERSION

```
#define FIRMATA_PROTOCOL_MINOR_VERSION firmata::PROTOCOL_MINOR_VERSION
```

Definition at line 32 of file FirmataDefines.h.

#### 27.19.1.16 I2C_CONFIG

```
#define I2C_CONFIG firmata::I2C_CONFIG
```

Definition at line 149 of file FirmataDefines.h.

#### 27.19.1.17 I2C_REPLY

```
#define I2C_REPLY firmata::I2C_REPLY
```

Definition at line 144 of file FirmataDefines.h.

#### 27.19.1.18 I2C_REQUEST

```
#define I2C_REQUEST firmata::I2C_REQUEST
```

Definition at line 139 of file FirmataDefines.h.

#### 27.19.1.19 MAX_DATA_BYTES

```
#define MAX_DATA_BYTES firmata::MAX_DATA_BYTES
```

Definition at line 38 of file FirmataDefines.h.

#### 27.19.1.20 ONEWIRE_DATA

```
#define ONEWIRE_DATA firmata::ONEWIRE_DATA
```

Definition at line 129 of file FirmataDefines.h.

#### 27.19.1.21 PIN_MODE_ANALOG

```
#define PIN_MODE_ANALOG firmata::PIN_MODE_ANALOG
```

Definition at line 226 of file FirmataDefines.h.

#### 27.19.1.22 PIN_MODE_ENCODER

```
#define PIN_MODE_ENCODER firmata::PIN_MODE_ENCODER
```

Definition at line 261 of file FirmataDefines.h.

#### 27.19.1.23 PIN_MODE_I2C

```
#define PIN_MODE_I2C firmata::PIN_MODE_I2C
```

Definition at line 246 of file FirmataDefines.h.

#### 27.19.1.24 PIN_MODE_IGNORE

```
#define PIN_MODE_IGNORE firmata::PIN_MODE_IGNORE
```

Definition at line 276 of file FirmataDefines.h.

#### 27.19.1.25 PIN_MODE_INPUT

```
#define PIN_MODE_INPUT firmata::PIN_MODE_INPUT
```

Definition at line 216 of file FirmataDefines.h.

#### 27.19.1.26 PIN_MODE_ONEWIRE

```
#define PIN_MODE_ONEWIRE firmata::PIN_MODE_ONEWIRE
```

Definition at line 251 of file FirmataDefines.h.

#### 27.19.1.27 PIN_MODE_OUTPUT

```
#define PIN_MODE_OUTPUT firmata::PIN_MODE_OUTPUT
```

Definition at line 221 of file FirmataDefines.h.

#### 27.19.1.28 PIN_MODE_PULLUP

```
#define PIN_MODE_PULLUP firmata::PIN_MODE_PULLUP
```

Definition at line 271 of file FirmataDefines.h.

#### 27.19.1.29 PIN_MODE_PWM

```
#define PIN_MODE_PWM firmata::PIN_MODE_PWM
```

Definition at line 231 of file FirmataDefines.h.

#### 27.19.1.30 PIN_MODE_SERIAL

```
#define PIN_MODE_SERIAL firmata::PIN_MODE_SERIAL
```

Definition at line 266 of file FirmataDefines.h.

#### 27.19.1.31 PIN_MODE_SERVO

```
#define PIN_MODE_SERVO firmata::PIN_MODE_SERVO
```

Definition at line 236 of file FirmataDefines.h.

#### 27.19.1.32 PIN_MODE_SHIFT

```
#define PIN_MODE_SHIFT firmata::PIN_MODE_SHIFT
```

Definition at line 241 of file FirmataDefines.h.

#### 27.19.1.33 PIN_MODE_STEPPER

```
#define PIN_MODE_STEPPER firmata::PIN_MODE_STEPPER
```

Definition at line 256 of file FirmataDefines.h.

#### 27.19.1.34 PIN_STATE_QUERY

```
#define PIN_STATE_QUERY firmata::PIN_STATE_QUERY
```

Definition at line 164 of file FirmataDefines.h.

#### 27.19.1.35 PIN_STATE_RESPONSE

```
#define PIN_STATE_RESPONSE firmata::PIN_STATE_RESPONSE
```

Definition at line 169 of file FirmataDefines.h.

#### 27.19.1.36 REPORT_ANALOG

```
#define REPORT_ANALOG firmata::REPORT_ANALOG
```

Definition at line 55 of file FirmataDefines.h.

#### 27.19.1.37 REPORT_DIGITAL

```
#define REPORT_DIGITAL firmata::REPORT_DIGITAL
```

Definition at line 60 of file FirmataDefines.h.

#### 27.19.1.38 REPORT_FIRMWARE

```
#define REPORT_FIRMWARE firmata::REPORT_FIRMWARE
```

Definition at line 154 of file FirmataDefines.h.

#### 27.19.1.39 REPORT_VERSION

```
#define REPORT_VERSION firmata::REPORT_VERSION
```

Definition at line 79 of file FirmataDefines.h.

#### 27.19.1.40 SAMPLING_INTERVAL

```
#define SAMPLING_INTERVAL firmata::SAMPLING_INTERVAL
```

Definition at line 194 of file FirmataDefines.h.

#### 27.19.1.41 SCHEDULER_DATA

```
#define SCHEDULER_DATA firmata::SCHEDULER_DATA
```

Definition at line 199 of file FirmataDefines.h.

#### 27.19.1.42 SERIAL_MESSAGE

```
#define SERIAL_MESSAGE firmata::SERIAL_DATA
```

Definition at line 104 of file FirmataDefines.h.

#### 27.19.1.43 SERVO_CONFIG

```
#define SERVO_CONFIG firmata::SERVO_CONFIG
```

Definition at line 114 of file FirmataDefines.h.

#### 27.19.1.44 SET_DIGITAL_PIN_VALUE

```
#define SET_DIGITAL_PIN_VALUE firmata::SET_DIGITAL_PIN_VALUE
```

Definition at line 72 of file FirmataDefines.h.

#### 27.19.1.45 SET_PIN_MODE

```
#define SET_PIN_MODE firmata::SET_PIN_MODE
```

Definition at line 67 of file FirmataDefines.h.

#### 27.19.1.46 SHIFT_DATA

```
#define SHIFT_DATA firmata::SHIFT_DATA
```

Definition at line 134 of file FirmataDefines.h.

#### 27.19.1.47 START_SYSEX

```
#define START_SYSEX firmata::START_SYSEX
```

Definition at line 91 of file FirmataDefines.h.

#### 27.19.1.48 STEPPER_DATA

```
#define STEPPER_DATA firmata::STEPPER_DATA
```

Definition at line 124 of file FirmataDefines.h.

#### 27.19.1.49 STRING_DATA

```
#define STRING_DATA firmata::STRING_DATA
```

Definition at line 119 of file FirmataDefines.h.

#### 27.19.1.50 SYSEX_NON_REALTIME

```
#define SYSEX_NON_REALTIME firmata::SYSEX_NON_REALTIME
```

Definition at line 204 of file FirmataDefines.h.

#### 27.19.1.51 SYSEX_REALTIME

```
#define SYSEX_REALTIME firmata::SYSEX_REALTIME
```

Definition at line 209 of file FirmataDefines.h.

#### 27.19.1.52 SYSTEM_RESET

```
#define SYSTEM_RESET firmata::SYSTEM_RESET
```

Definition at line 84 of file FirmataDefines.h.

#### 27.19.1.53 TOTAL_PIN_MODES

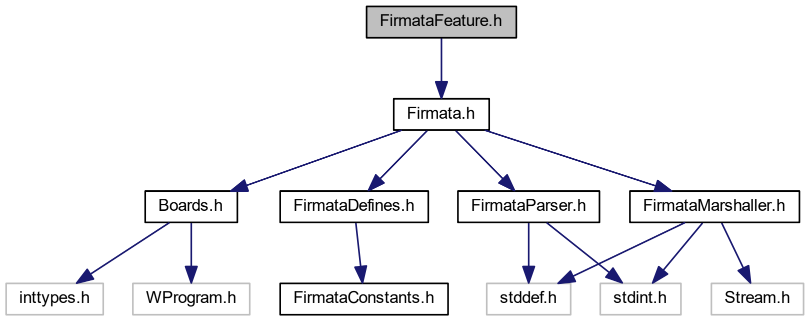
```
#define TOTAL_PIN_MODES firmata::TOTAL_PIN_MODES
```

Definition at line 281 of file FirmataDefines.h.

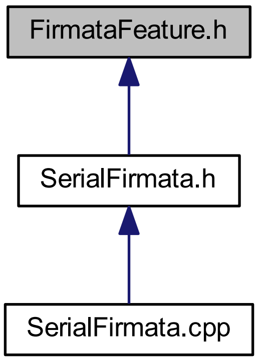
## 27.20 FirmataFeature.h File Reference

```
#include <Firmata.h>
```

Include dependency graph for FirmataFeature.h:



This graph shows which files directly or indirectly include this file:



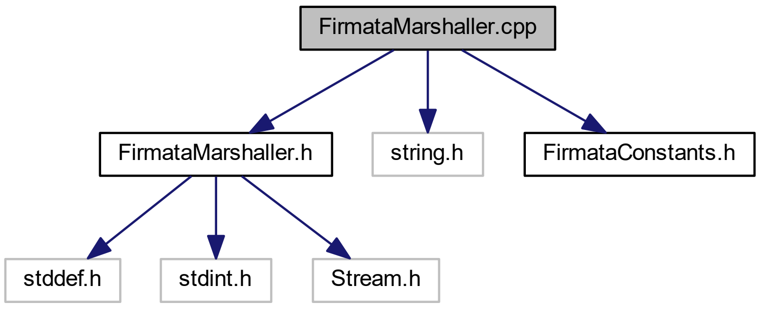
### Data Structures

- class FirmataFeature

## 27.21 FirmataMarshaller.cpp File Reference

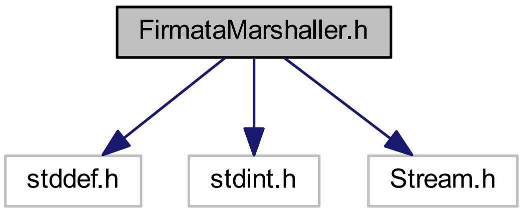
```
#include "FirmataMarshaller.h"
#include <string.h>
```

```
#include "FirmataConstants.h"
```
Include dependency graph for FirmataMarshaller.cpp:

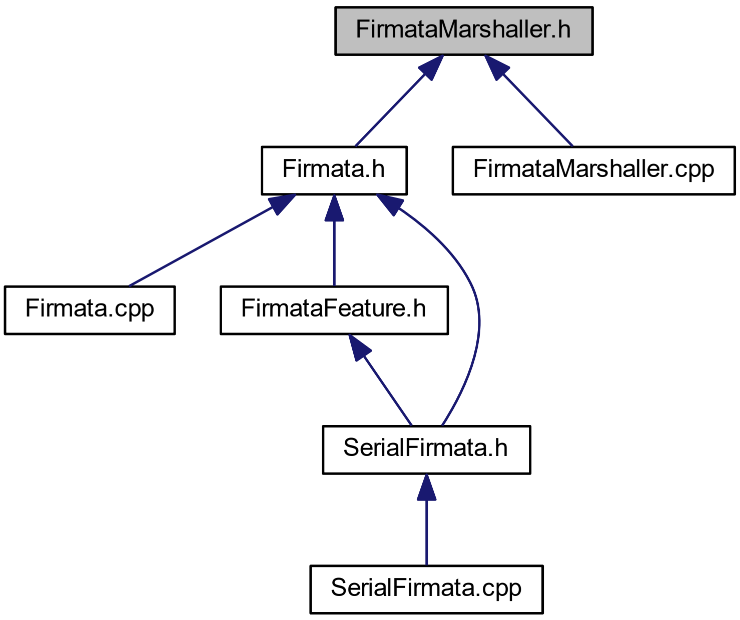


## 27.22 FirmataMarshaller.h File Reference

```
#include <stddef.h>
#include <stdint.h>
#include <Stream.h>
```
Include dependency graph for FirmataMarshaller.h:



This graph shows which files directly or indirectly include this file:



### Data Structures

- class firmata::FirmataMarshaller

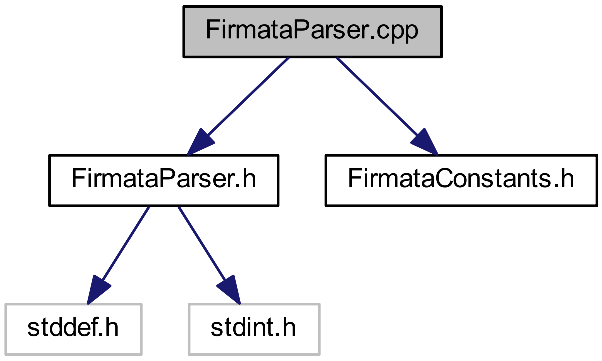
### Namespaces

- firmata

## 27.23 FirmataParser.cpp File Reference

```
#include "FirmataParser.h"
#include "FirmataConstants.h"
```

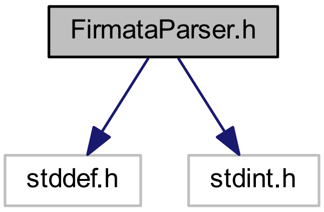
Include dependency graph for FirmataParser.cpp:



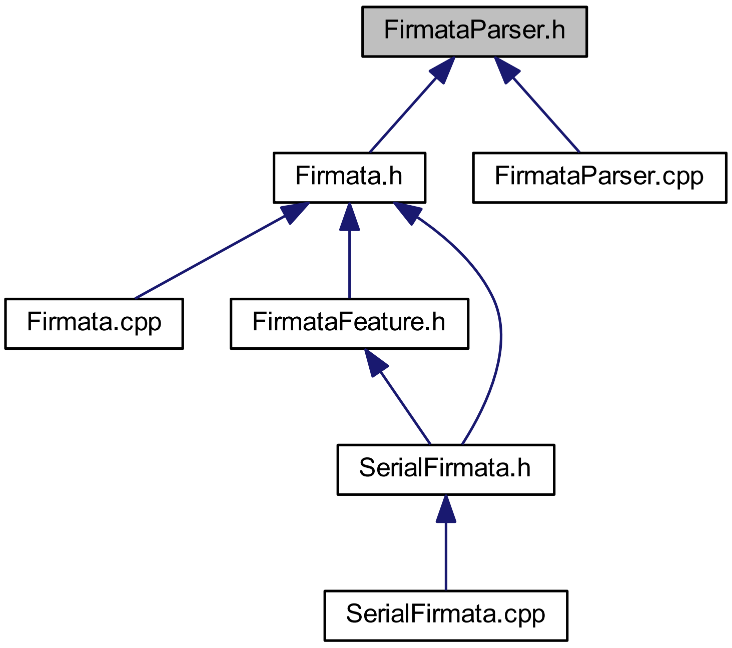
## 27.24 FirmataParser.h File Reference

```
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for FirmataParser.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class firmata::FirmataParser

## Namespaces

- firmata

## 27.25 i2c.md File Reference

## 27.26 onewire.md File Reference

## 27.27 pingroups-proposal.md File Reference

## 27.28 protocol.md File Reference

## 27.29 rcswitch-proposal.md File Reference

## 27.30 readme.md File Reference

## 27.31 readme.md File Reference

## 27.32 README.md File Reference

## 27.33 README.md File Reference

## 27.34 revisions.md File Reference

## 27.35 scheduler.md File Reference

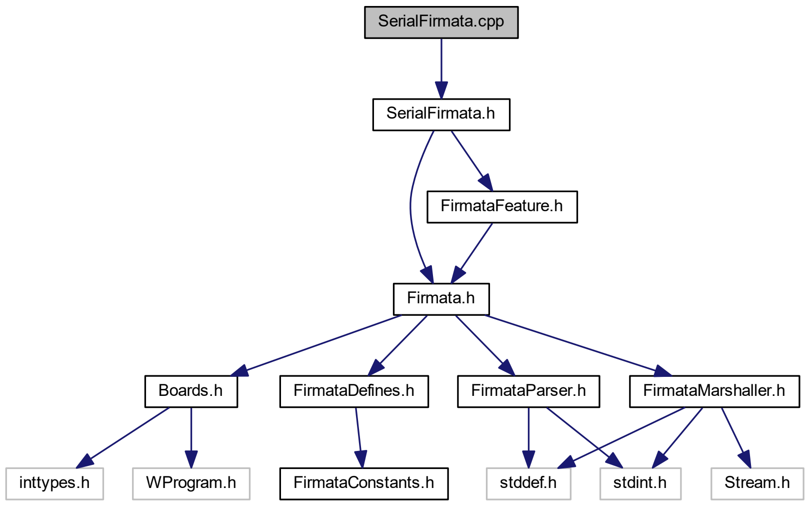
## 27.36 serial-1.0.md File Reference

## 27.37 serial-2.0-proposal.md File Reference

## 27.38 SerialFirmata.cpp File Reference

```
#include "SerialFirmata.h"
```

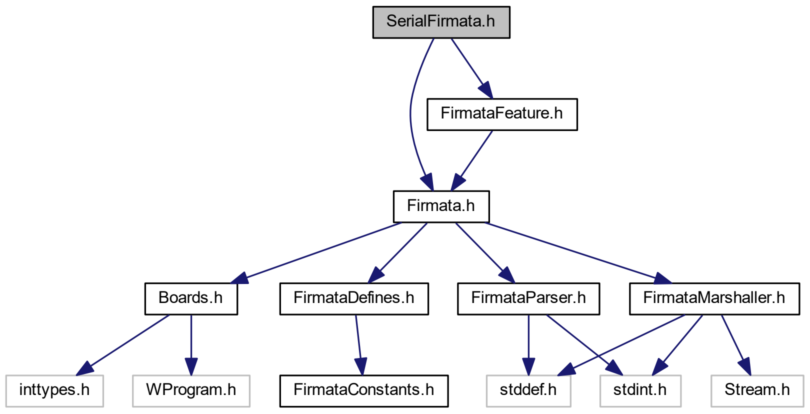
Include dependency graph for SerialFirmata.cpp:



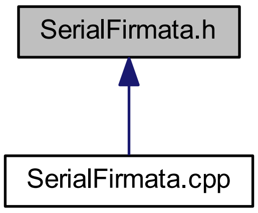
## 27.39 SerialFirmata.h File Reference

```
#include <Firmata.h>
#include "FirmataFeature.h"
```

Include dependency graph for SerialFirmata.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class SerialFirmata

## Macros

- #define FIRMATA_SERIAL_FEATURE
- #define HW_SERIAL0 0x00
- #define HW_SERIAL1 0x01
- #define HW_SERIAL2 0x02
- #define HW_SERIAL3 0x03
- #define HW_SERIAL4 0x04
- #define HW_SERIAL5 0x05
- #define HW_SERIAL6 0x06
- #define SW_SERIAL0 0x08
- #define SW_SERIAL1 0x09
- #define SW_SERIAL2 0x0A
- #define SW_SERIAL3 0x0B
- #define SERIAL_PORT_ID_MASK 0x0F
- #define MAX_SERIAL_PORTS 8
- #define SERIAL_READ_ARR_LEN 12
- #define RES_RX1 0x02
- #define RES_TX1 0x03
- #define RES_RX2 0x04
- #define RES_TX2 0x05
- #define RES_RX3 0x06
- #define RES_TX3 0x07
- #define RES_RX4 0x08
- #define RES_TX4 0x09
- #define RES_RX5 0x0a
- #define RES_TX5 0x0b
- #define RES_RX6 0x0c
- #define RES_TX6 0x0d
- #define SERIAL_CONFIG 0x10
- #define SERIAL_WRITE 0x20
- #define SERIAL_READ 0x30

- #define SERIAL_REPLY 0x40
- #define SERIAL_CLOSE 0x50
- #define SERIAL_FLUSH 0x60
- #define SERIAL_LISTEN 0x70
- #define SERIAL_READ_CONTINUOUSLY 0x00
- #define SERIAL_STOP_READING 0x01
- #define SERIAL_MODE_MASK 0xF0

## 27.39.1 Macro Definition Documentation

#### 27.39.1.1 FIRMATA_SERIAL_FEATURE

```
#define FIRMATA_SERIAL_FEATURE
```

Definition at line 33 of file SerialFirmata.h.

#### 27.39.1.2 HW_SERIAL0

```
#define HW_SERIAL0 0x00
```

Definition at line 36 of file SerialFirmata.h.

#### 27.39.1.3 HW_SERIAL1

```
#define HW_SERIAL1 0x01
```

Definition at line 37 of file SerialFirmata.h.

#### 27.39.1.4 HW_SERIAL2

```
#define HW_SERIAL2 0x02
```

Definition at line 38 of file SerialFirmata.h.

#### 27.39.1.5 HW_SERIAL3

```
#define HW_SERIAL3 0x03
```

Definition at line 39 of file SerialFirmata.h.

#### 27.39.1.6 HW_SERIAL4

```
#define HW_SERIAL4 0x04
```

Definition at line 40 of file SerialFirmata.h.

#### 27.39.1.7 HW_SERIAL5

```
#define HW_SERIAL5 0x05
```

Definition at line 41 of file SerialFirmata.h.

#### 27.39.1.8 HW_SERIAL6

```
#define HW_SERIAL6 0x06
```

Definition at line 42 of file SerialFirmata.h.

#### 27.39.1.9 MAX_SERIAL_PORTS

```
#define MAX_SERIAL_PORTS 8
```

Definition at line 52 of file SerialFirmata.h.

#### 27.39.1.10 RES_RX1

```
#define RES_RX1 0x02
```

Definition at line 56 of file SerialFirmata.h.

#### 27.39.1.11 RES_RX2

```
#define RES_RX2 0x04
```

Definition at line 58 of file SerialFirmata.h.

#### 27.39.1.12 RES_RX3

```
#define RES_RX3 0x06
```

Definition at line 60 of file SerialFirmata.h.

#### 27.39.1.13 RES_RX4

```
#define RES_RX4 0x08
```

Definition at line 62 of file SerialFirmata.h.

#### 27.39.1.14 RES_RX5

```
#define RES_RX5 0x0a
```

Definition at line 64 of file SerialFirmata.h.

#### 27.39.1.15 RES_RX6

```
#define RES_RX6 0x0c
```

Definition at line 66 of file SerialFirmata.h.

#### 27.39.1.16 RES_TX1

```
#define RES_TX1 0x03
```

Definition at line 57 of file SerialFirmata.h.

#### 27.39.1.17 RES_TX2

```
#define RES_TX2 0x05
```

Definition at line 59 of file SerialFirmata.h.

#### 27.39.1.18 RES_TX3

```
#define RES_TX3 0x07
```

Definition at line 61 of file SerialFirmata.h.

#### 27.39.1.19 RES_TX4

```
#define RES_TX4 0x09
```

Definition at line 63 of file SerialFirmata.h.

#### 27.39.1.20 RES_TX5

```
#define RES_TX5 0x0b
```

Definition at line 65 of file SerialFirmata.h.

#### 27.39.1.21 RES_TX6

```
#define RES_TX6 0x0d
```

Definition at line 67 of file SerialFirmata.h.

#### 27.39.1.22 SERIAL_CLOSE

```
#define SERIAL_CLOSE 0x50
```

Definition at line 74 of file SerialFirmata.h.

#### 27.39.1.23 SERIAL_CONFIG

```
#define SERIAL_CONFIG 0x10
```

Definition at line 70 of file SerialFirmata.h.

#### 27.39.1.24 SERIAL_FLUSH

```
#define SERIAL_FLUSH 0x60
```

Definition at line 75 of file SerialFirmata.h.

#### 27.39.1.25 SERIAL_LISTEN

```
#define SERIAL_LISTEN 0x70
```

Definition at line 76 of file SerialFirmata.h.

#### 27.39.1.26 SERIAL_MODE_MASK

```
#define SERIAL_MODE_MASK 0xF0
```

Definition at line 81 of file SerialFirmata.h.

#### 27.39.1.27 SERIAL_PORT_ID_MASK

```
#define SERIAL_PORT_ID_MASK 0x0F
```

Definition at line 51 of file SerialFirmata.h.

#### 27.39.1.28 SERIAL_READ

```
#define SERIAL_READ 0x30
```

Definition at line 72 of file SerialFirmata.h.

#### 27.39.1.29 SERIAL_READ_ARR_LEN

```
#define SERIAL_READ_ARR_LEN 12
```

Definition at line 53 of file SerialFirmata.h.

#### 27.39.1.30 SERIAL_READ_CONTINUOUSLY

```
#define SERIAL_READ_CONTINUOUSLY 0x00
```

Definition at line 79 of file SerialFirmata.h.

#### 27.39.1.31 SERIAL_REPLY

```
#define SERIAL_REPLY 0x40
```

Definition at line 73 of file SerialFirmata.h.

#### 27.39.1.32 SERIAL_STOP_READING

```
#define SERIAL_STOP_READING 0x01
```

Definition at line 80 of file SerialFirmata.h.

#### 27.39.1.33 SERIAL_WRITE

```
#define SERIAL_WRITE 0x20
```

Definition at line 71 of file SerialFirmata.h.

#### 27.39.1.34 SW_SERIAL0

```
#define SW_SERIAL0 0x08
```

Definition at line 45 of file SerialFirmata.h.

#### 27.39.1.35 SW_SERIAL1

```
#define SW_SERIAL1 0x09
```

Definition at line 46 of file SerialFirmata.h.

#### 27.39.1.36 SW_SERIAL2

```
#define SW_SERIAL2 0x0A
```

Definition at line 47 of file SerialFirmata.h.

#### 27.39.1.37 SW_SERIAL3

```
#define SW_SERIAL3 0x0B
```

Definition at line 48 of file SerialFirmata.h.

## 27.40 servos.md File Reference

## 27.41 shift-proposal.md File Reference

## 27.42 spi-proposal.md File Reference

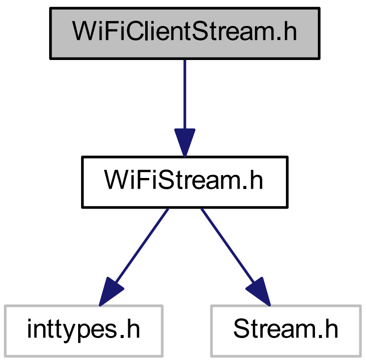
## 27.43 stepper-legacy.md File Reference

## 27.44 tone-proposal.md File Reference

## 27.45 WiFiClientStream.h File Reference

```
#include "WiFiStream.h"
```

Include dependency graph for WiFiClientStream.h:



### Data Structures

- class WiFiClientStream

### Macros

- #define MILLIS_RECONNECT 5000

### 27.45.1 Macro Definition Documentation

#### 27.45.1.1 MILLIS_RECONNECT

```
#define MILLIS_RECONNECT 5000
```

Definition at line 31 of file WiFiClientStream.h.

## 27.46 wifiConfig.h File Reference

### Macros

- #define SERVER_PORT 3030
- #define WIFI_WPA_SECURITY

### Functions

- WiFiServerStream stream (SERVER_PORT)

### Variables

- char ssid [ ] = "your_network_name"
- char wpa_passphrase [ ] = "your_wpa_passphrase"

### 27.46.1 Macro Definition Documentation

#### 27.46.1.1 SERVER_PORT

```
#define SERVER_PORT 3030
```

Definition at line 168 of file wifiConfig.h.

#### 27.46.1.2 WIFI_WPA_SECURITY

```
#define WIFI_WPA_SECURITY
```

Definition at line 183 of file wifiConfig.h.

### 27.46.2 Function Documentation

#### 27.46.2.1 stream()

```
WiFiServerStream stream (
             SERVER_PORT   )
```

### 27.46.3 Variable Documentation

#### 27.46.3.1 ssid

```
char ssid[] = "your_network_name"
```

Definition at line 154 of file wifiConfig.h.

#### 27.46.3.2 wpa_passphrase

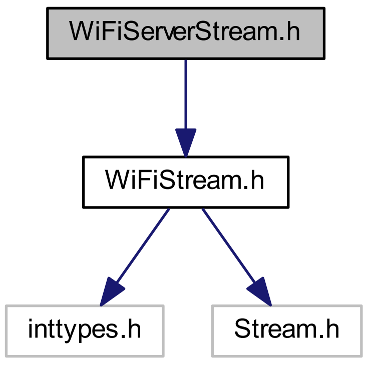
```
char wpa_passphrase[] = "your_wpa_passphrase"
```

Definition at line 186 of file wifiConfig.h.

## 27.47 WiFiServerStream.h File Reference

```
#include "WiFiStream.h"
```

Include dependency graph for WiFiServerStream.h:



### Data Structures

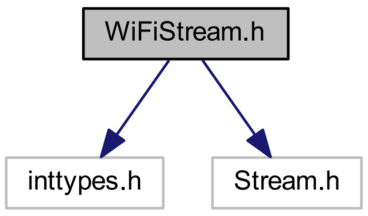
- class WiFiServerStream

## 27.48 WiFiStream.cpp File Reference

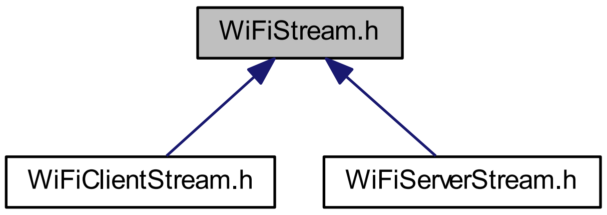
## 27.49 WiFiStream.h File Reference

```
#include <inttypes.h>
#include <Stream.h>
```

Include dependency graph for WiFiStream.h:



This graph shows which files directly or indirectly include this file:



### Data Structures

- class WiFiStream

### Macros

- #define HOST_CONNECTION_DISCONNECTED 0
- #define HOST_CONNECTION_CONNECTED 1

## Typedefs

- typedef void(∗ hostConnectionCallbackFunction) (byte)

### 27.49.1 Macro Definition Documentation

#### 27.49.1.1 HOST_CONNECTION_CONNECTED

```
#define HOST_CONNECTION_CONNECTED 1
```

Definition at line 28 of file WiFiStream.h.

#### 27.49.1.2 HOST_CONNECTION_DISCONNECTED

```
#define HOST_CONNECTION_DISCONNECTED 0
```

Definition at line 27 of file WiFiStream.h.

### 27.49.2 Typedef Documentation

#### 27.49.2.1 hostConnectionCallbackFunction

```
typedef void(* hostConnectionCallbackFunction) (byte)
```

Definition at line 32 of file WiFiStream.h.

# Index