arduino-simple-rpc Documentation

Release latest

Jun 11, 2019

Contents:

This library provides a simple way to interface to Arduino functions exported with the simpleRPC protocol. The exported method definitions are communicated to the host, which is then able to generate an API interface using this library.

Features:

- User friendly API library.
- Command line interface (CLI) for method discovery and testing.
- Function and parameter names are defined on the Arduino.
- API documentation is defined on the Arduino.
- Support for disconnecting and reconnecting.

Please see ReadTheDocs for the latest documentation.

CHAPTER 1

Quick start

Export any function e.g., `digitalRead()` and `digitalWrite()` on the Arduino, these functions will show up as member functions of the `Interface` class instance.

First, we make an `Interface` class instance and tell it to connect to the serial device `/dev/ttyACM0`.

```
>>> from simple_rpc import Interface
>>>
>>> interface = Interface('/dev/ttyACM0')
```

We can use the built-in `help()` function to see the API documentation of any exported method.

```
>>> help(interface.digital_read)
Help on method digital_read:

digital_read(pin) method of simple_rpc.simple_rpc.Interface instance
    Read digital pin.

    :arg int pin: Pin number.

    :returns int: Pin value.
```

All exposed methods can be called like any other class method.

```
>>> interface.digital_read(8)          # Read from pin 8.
0
>>> interface.digital_write(13, True) # Turn LED on.
```

CHAPTER 2

Further reading

For more information about the host library and other interfaces, please see the Usage and Library sections.

2.1 Introduction

This Python library provides a simple way to interface to Arduino functions exported with the simpleRPC protocol.

For more background information and the reasons that led to this project, see the motivation section of the device library documentation.

This project serves as a reference implementation for clients using the simpleRPC protocol.

2.2 Installation

The software is distributed via PyPI, it can be installed with `pip`:

```
pip install arduino-simple-rpc
```

2.2.1 From source

The source is hosted on GitHub, to install the latest development version, use the following commands.

```
git clone https://github.com/jfjlaros/arduino-simple-rpc.git
cd arduino-simple-rpc
pip install .
```

2.3 Usage

The command line interface can be useful for method discovery and testing purposes. It currently has two subcommands: `list`, which shows a list of available methods and `call` for calling methods. For more information, use the `-h` option.

```
simple_rpc -h
```

2.3.1 Basic usage

If the Arduino has exposed the functions `inc` and `set_led` like in the example given in the device library documentation, the `list` subcommand will show the following.

```
$ simple_rpc list
Available methods:


inc a
    Increment a value.

    int a: Value.

    returns int: a + 1.


set_led brightness
    Set LED brightness.

    int brightness: Brightness.
```

Any of these methods can be called by using the `call` subcommand.

```
$ simple_rpc call inc 1
2
```

2.3.2 Complex objects

Complex objects are passed on the command line interface as a JSON string. Binary encoding and decoding is taken care of by the CLI.

```
$ simple_rpc call vector '[1, 2, 3, 4]'
[1.40, 2.40, 3.40, 4.40]

$ simple_rpc call object '["a", [10, "b"]]'
["b", [11, "c"]]
```

2.4 Library

The API library provides the `Interface` class. A class instance is made by passing the path to a serial device to the constructor.

```
>>> from simple_rpc import Interface
>>>
>>> interface = Interface('/dev/ttyACM0')
```

Every exported method will show up as a class method of the `interface` class instance. These methods can be used like any normal class methods. Alternatively, the exported methods can be called by name using the `call_method()` function.

The constructor takes the following parameters.

Table 1: Constructor parameters.

name	optional	description
`device`	no	Serial device name.
`baudrate`	yes	Baud rate.
`wait`	yes	Time in seconds before communication starts.
`autoconnect`	yes	Automatically connect.

The following standard methods are available.

Table 2: Class methods.

name	description
`open()`	Connect to serial device.
`close()`	Disconnect from serial device.
`is_open()`	Query serial device state.
`call_method()`	Execute a method.

If the connection should not be made instantly, the `autoconnect` parameter can be used in combination with the `open()` function.

```
>>> interface = Interface('/dev/ttyACM0', autoconnect=False)
>>> # Do something.
>>> interface.open()
```

The connection state can be queried using the `is_open()` function and it can be closed using the `close()` function.

```
>>> if interface.is_open():
>>>     interface.close()
```

Additionally, the `with` statement is supported for easy opening and closing.

```
>>> with Interface('/dev/ttyACM0') as interface:
>>>     interface.ping(10)
```

The class instance has a public member variable named `methods` which contains the definitions of the exported methods.

```
>>> interface.methods.keys()
dict_keys(['inc', 'set_led'])
>>> interface.methods['inc']
{
  'return': {
    'doc': 'a + 1.',
    'fmt': b'h',
    'typename': 'int'},
```

(continues on next page)

(continued from previous page)

```
  'doc': 'Increment a value.',
  'name': 'inc',
  'index': 2,
  'parameters': [
    {
      'doc': 'Value.',
      'name': 'a',
      'fmt': b'h',
      'typename': 'int'
    }
  ]
}
```

2.4.1 Basic usage

In the example given in the device library documentation, the `inc` method is exported, which is now present as a class method of the `Interface` class instance.

```
>>> interface.inc(1)
2
```

Alternatively, the exported method can be called using the `call_mathod()` function.

```
>>> interface.call_method('inc', 1)
2
```

To get more information about this class method, the built-in `help()` function can be used.

```
>>> help(interface.inc)
Help on method inc:

inc(a) method of simple_rpc.simple_rpc.Interface instance
    Increment a value.

    :arg int a: Value.

    :returns int: a + 1.
```

Note that strings should be encoded as `bytes` objects. If, for example, we have a function named `test` that takes a string as parameter, we should call this function as follows.

```
>>> interface.test(b'hello world')
```

2.4.2 Complex objects

Some methods may have complex objects like Tuples, Objects or Vectors as parameters or return type.

In the following example, we call a method that takes a Vector of integers and returns a Vector of floats.

```
>>> interface.vector([1, 2, 3, 4])
[1.40, 2.40, 3.40, 4.40]
```

In this example, we call a method that takes an Object containing a byte and an other Object. A similar Object is returned.

```
>>> interface.object((b'a', (10, b'b')))
(b'b', (11, b'c'))
```

2.5 Contributors

- Jeroen F.J. Laros <jlaros@fixedpoint.nl> (Original author, maintainer)

Find out who contributed:

```
git shortlog -s -e
```