socketscpi Documentation

Release 2023.06.0

Morgan Allison

Jun 13, 2023

Contents:

1 socketscpi **1**
- 1.1 Features . . . 1

2 Installation **3**
- 2.1 Stable release . . . 3
- 2.2 From sources . . . 3

3 Usage **5**

4 SocketInstrument **7**
- 4.1 **close** . . . 7
- 4.2 **write** . . . 8
- 4.3 **read** . . . 8
- 4.4 **query** . . . 8
- 4.5 **err_check** . . . 8
- 4.6 **query_binary_values** . . . 9
- 4.7 **write_binary_values** . . . 9

5 Contributing **11**
- 5.1 Types of Contributions . . . 11
- 5.2 Get Started! . . . 12
- 5.3 Pull Request Guidelines . . . 13
- 5.4 Tips . . . 13
- 5.5 Deploying . . . 13

6 Credits **15**
- 6.1 Development Lead . . . 15
- 6.2 Contributors . . . 15

7 History **17**
- 7.1 0.0.1 (2019-01-24) . . . 17
- 7.2 0.0.4 (2019-04-26) . . . 17
- 7.3 2020.04.0 (2020-04-15) . . . 17
- 7.4 2020.05.0 (2020-05-13) . . . 17
- 7.5 2022.08.0 (2022-08-11) . . . 17
- 7.6 2023.04.0 (2023-04-17) . . . 18
- 7.7 2023.06.0 (2023-06-13) . . . 18

8 Indices and tables 19

CHAPTER 1

socketscpi

Tired of troubleshooting VISA connections, conflicts, and incompatibilities?

Need the fastest communication possible with your test equipment?

Try socketscpi: a robust and easy-to-use SCPI interface for electronic test and measurement equipment.

Socketscpi is a wrapper for Python's socket module. This removes the requirement for VISA and improves data transfer speed over the older VXI-11 protocol.

1.1 Features

- Written using the socket module for fast communication
- Implements write, read, query, binary block read, binary block write
- Free software: MIT License
- Documentation: https://socketscpi.readthedocs.io/en/latest/index.html

CHAPTER 2

Installation

2.1 Stable release

To install socketscpi, run this command in your terminal:

```
$ pip install socketscpi
```

This is the preferred method to install socketscpi, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for socketscpi can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/morgan-at-keysight/socketscpi
```

Or download the tarball:

```
$ curl  -OL https://github.com/morgan-at-keysight/socketscpi/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use socketscpi in a project:

```
import socketscpi
```

To create an instrument object, do something like this:

```
ipAddress = '192.168.1.123'
instrument = socketscpi.SocketInstrument(ipAddress)
```

To send SCPI commands and queries to the instrument, do something like this:

```
instrument.write('*rst')
instrument.query('*opc?')
```

To check for and print out errors, do something like this:

```
try:
    instrument.err_check()
except socketscpi.SockInstError as e:
    print(str(e))
```

When you're finished communicating with your instrument, close it gracefully like this:

```
instrument.close()
```

CHAPTER 4

SocketInstrument

```
socketscpi.SocketInstrument(host, port=5025, timeout=10, noDelay=True,␣
↪globalErrCheck=False, verboseErrCheck=True)
```

Class constructor that connects to the test equipment and returns a SocketInstrument object that can be used to communicate with the equipment.

Arguments

- `host (string)`: Instrument host IP address. Argument is a string containing a valid IP address.
- `port (int)`: Port used by the instrument to facilitate socket communication (Keysight equipment uses port 5025 by default).
- `timeout (int)`: Timeout in seconds. This is how long the instrument will wait before sending a timeout error in response to a command or query. Argument is an int. Default is `10`.
- `noDelay (bool)`: True turns on the TCP_NODELAY flag, which sends data immediately without concatenating multiple packets together. Just leave this alone.
- `globalErrCheck (bool)`: Determines if error checking will be done automatically after calling class methods.
- `verboseErrCheck (bool)`: Determines if verbose error checking will be attempted.

Returns

- `socketscpi.SocketInstrument`: Instrument object to be used for communication and control.

4.1 close

```
SocketInstrument.close()
```

Gracefully closes socket connection.

Arguments

- None

Returns

- None

4.2 write

```
SocketInstrument.write(cmd)
```

Writes a command to the instrument.

Arguments

- `cmd (string)`: Documented SCPI command to be sent to the instrument.

Returns

- None

4.3 read

```
SocketInstrument.read()
```

Reads the output buffer of the instrument.

Arguments

- None

Returns

- `(string)`: Contents of the instrument's output buffer.

4.4 query

```
SocketInstrument.query(cmd)
```

Sends query to instrument and reads the output buffer immediately afterward.

Arguments

- `cmd (string)`: Documented SCPI query to be sent to instrument (should end in a "?" character).

Returns

- `(string)` Response from instrument's output buffer as a latin_1-encoded string.

4.5 err_check

```
SocketInstrument.err_check()
```

Prints out all errors and clears error queue. Raises SockInstError with the info of the error encountered.

Arguments

- None

Returns

- None

4.6 query_binary_values

```
SocketInstrument.query_binary_values(cmd, datatype='b')
```

Sends a query and parses response in IEEE 488.2 binary block format.

Arguments

- `cmd (string)`: Documented SCPI query that causes the instrument to return a binary block.
- `datatype (string)`: Data type for the returned data. Uses the same naming convention used by Python's built-in `struct` module. Generally, test equipment includes a command to configure the data type of binary blocks, and the instrument's data type should match the data type used here. Default is `'b'`, which specifies a signed 8 bit integer.

Returns

- `(NumPy ndarray)` Array containing the data from the instrument buffer.

4.7 write_binary_values

```
SocketInstrument.write_binary_values(cmd, data)
```

Sends a command and payload data in IEEE 488.2 binary block format.

Arguments

- `cmd (string)`: SCPI command used to send data to instrument as a binary block.
- `data (NumPy ndarray)`: Data to be sent to the instrument. Refer to the documentation of the SCPI command being used for correct argument formatting.
- `esr (bool)`: Determines whether to append an ESR query to the end of the binary block write for error checking purposes.

Returns

- None

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/morgan-at-keysight/socketscpi/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

socketscpi could always use more documentation, whether as part of the official socketscpi docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/morgan-at-keysight/socketscpi/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *socketscpi* for local development.

1. Fork the *socketscpi* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/socketscpi.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv socketscpi
$ cd socketscpi/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 socketscpi tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/morgan-at-keysight/socketscpi/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_socketscpi
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Morgan Allison <morgan.j.allison@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.0.1 (2019-01-24)

- First release on PyPI.

7.2 0.0.4 (2019-04-26)

- Updated syntax for binblockread to mimic that of PyVISA. Created documentation.

7.3 2020.04.0 (2020-04-15)

- Added a `.read()` method. Wrote test scripts to verify performance. Overhauled documentation. Switched to calendar-style versioning.

7.4 2020.05.0 (2020-05-13)

- Adjusted the error checking for the `.query()` method to account for SCPI queries that require additional arguments.

7.5 2022.08.0 (2022-08-11)

- Renamed `binblockwrite()`, `binblockread()`, and `disconnect()` to `write_binary_values()`, `read_binary_values()`, and `close()`, respectively, to match the function calls in PyVISA.

7.6 2023.04.0 (2023-04-17)

- Added error checking syntax for UXR scopes. Added an argument in the `SocketInstrument` constructor to allow user to decide if verbose error checking will be attempted.

7.7 2023.06.0 (2023-06-13)

- Relaxed error checking to account for different "No error" responses from different instrument vendors. Updated comments.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search