Gearthonic Documentation

Release 0.1.1

Timo Steidle

July 13, 2016

Contents

A simple client to the XML RPC API of Homegear.

Look at the documentation for detailed information.

CHAPTER 1

Quickstart

Install *gearthonic* via pip:

```
pip install gearthonic
```

Initialise the client:

```
from gearthonic import GearClient
gc = GearClient('192.168.1.100', 2001, secure=True)  # 2001 is the default port of the Homegear serve
```

Use the predefined methods to make requests to the XML RPC API:

```
gc.device.list_methods()
gc.device.get_value(1, 4, 'ACTUAL_TEMPERATURE')
```

Alternatively you can call any method directly via the client:

```
gc.getValue(1, 4, 'ACTUAL_TEMPERATURE')
```

CHAPTER 2

Contents:

2.1 Installation

At the command line either via easy_install or pip:

```
$ easy_install gearthonic
$ pip install gearthonic
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv gearthonic
$ pip install gearthonic
```

2.2 Usage

To use Gearthonic in a project:

```
from gearthonic import GearClient
# Initialise the client with the host and port of your homegear server
gc = GearClient('192.168.1.100', 2001, secure=True)
# Now you can make your requests
gc.device.list_methods()
gc.device.get_value(1, 4, 'ACTUAL_TEMPERATURE')
```

The default port of the Homegear XML RPC server is `2001`.

All methods are already implemented within the client to make it easy to use. You don't have to lookup all methods and their parameters, just look at the code or see Method Reference.

Alternatively you can call any method directly via the client or use the generic ''call''-method:

```
gc.getValue(1, 4, 'ACTUAL_TEMPERATURE')
gc.call('system.listDevices')
gc.call('getValue', 1, 4, 'ACTUAL_TEMPERATURE')
```

A full list of all available methods provided by the XML RPC server can be found in the wiki of the Homegear project.

2.2.1 Additional information

- Documentation of all available data endpoints for all devices

- wiki of the Homegear project

2.2.2 Security

If you set `secure=True` while initialising the GearClient, the client tries to establish a secure connection to the server via SSL. It's highly recommended to use SSL for the network traffic. Otherwise the communication is not encrypted.

2.3 API

2.3.1 API Reference

class `gearthonic.`**`GearClient`**(*host*, *port*, *secure=False*)

Client for the XML RPC API provided by Homegear.

`call`(*method_name*, **args*, ***kwargs*)

Call the given method using the ServerProxy.

Parameters

- **`method_name`** – Method to be called
- **`args`** – Arguments passed through
- **`kwargs`** – Keyword arguments passed through

Returns Return value of the XML RPC method

`register_server`(*uri*, *name*)

Register a XmlRPC-Server for callbacks

Parameters

- **`uri`** – URI of the server
- **`name`** – Server name

Returns

2.3.2 Method Reference

System methods

class `gearthonic.methods.`**`SystemMethodsCollection`**(*caller*)

All XML RPC server provide a set of standard methods.

`get_capabilities`()

Lists server's XML RPC capabilities.

Example output:

```
{
    'xmlrpc': {'specUrl': 'http://example.com', 'specVersion': 1}
    'faults_interop': {'specUrl': 'http://example2.com', 'specVersion': 101}
    'introspection': {'specUrl': 'http://example3.com', 'specVersion': 42}
}
```

Returns A dict containing all information

Return type dict

list_methods()
Lists servers's XML RPC methods.

Returns A list of available methods

Return type list

method_help(*method_name*)
Returns the description of a method.

Parameters **method_name** (*str*) – The name of the method

Returns The description of the method

Return type str

method_signature(*method_name*)
Returns the signature of a method.

Parameters **method_name** (*str*) – The name of the method

Returns The signature of the method

Return type list

multicall(*methods*)
Call multiple methods at once.

Example list of `methods`:

```
[
    {'methodName': 'getValue', 'params': [13, 4, 'TEMPERATURE']},
    {'methodName': 'getValue', 'params': [3, 3, 'HUMIDITY']}
]
```

Return value of the multicall:

```
[22.0, 58]
```

Parameters **methods** (*list*) – A list of methods and their parameters

Returns A list of method responses.

Return type list

Device methods

class gearthonic.methods.**DeviceMethodsCollection**(*caller*)
All device related methods.

get_value(*peer_id*, *channel*, *key*, *request_from_device=False*, *asynchronous=False*)
Return the value of the device, specified by channel and key (parameterName).

Per default the value is read from the local cache of Homegear. If the value should be read from the device, use `request_from_device`. If the value should be read from the device, this can be done asynchronously. The method returns immediately and doesn't wait for the current value. The value will be sent as an event as soon as it's returned by the device.

Error codes:

•Returns `-2` when the device or channel is unknown

•Returns `-5` when the key (parameter) is unknown

Parameters

- **peer_id** (*int*) – ID of the device
- **channel** (*int*) – Channel of the device to get the value for
- **key** (*str*) – Name of the parameter to get the value for
- **request_from_device** (*bool*) – If true value is read from the device
- **asynchronous** (*bool*) – If true value is read asynchronously

Returns The value of the parameter or error code

Return type unknown

list_devices()
Return a list of devices.

Returns List of devices

Return type list

set_value(*peer_id*, *channel*, *key*, *value*)
Set the value of the device, specified by channel and key (parameterName).

Parameters

- **peer_id** (*int*) – ID of the device
- **channel** (*int*) – Channel of the device to set the value for
- **key** (*str*) – Name of the parameter to get the value for
- **value** (*unknown*) – The value to set

Returns

- `None` on success
- `-2` when the device or channel is unknown
- `-5` when the key (parameter) is unknown
- `-100` when the device did not respond
- `-101` when the device returns an error

2.4 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:

2.4.1 Types of Contributions

Report Bugs

Report bugs at https://gitlab.com/wumpitz/gearthonic/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.

Fix Bugs

Look through the Gitlab issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the Gitlab issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

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

Submit Feedback

The best way to send feedback is to file an issue at https://gitlab.com/wumpitz/gearthonic/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 :)

2.4.2 Get Started!

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

1. Fork the *gearthonic* repo on Gitlab.
2. Clone your fork locally:

```
$ git clone git@gitlab.com:your_name_here/gearthonic.git
```

3. Create a branch for local development:

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

Now you can make your changes locally.

4. When you're done making changes, check that your changes pass style and unit tests, including testing other Python versions with tox:

```
$ tox
```

To get tox, just pip install it.

5. Commit your changes and push your branch to Gitlab:

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

6. Submit a pull request through the Gitlab website.

2.4.3 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 and 3.5, and for PyPy. Run the `tox` command and make sure that the tests pass for all supported Python versions.

2.4.4 Tips

To run a subset of tests:

```
$ py.test test/test_gearthonic.py
```

2.5 Changelog

2.5.1 0.1.1 (2016-07-13)

- Added documentation.

2.5.2 0.1.0 (2016-07-13)

- Initial release to pypi.

CHAPTER 3

Feedback

If you have any suggestions or questions about **gearthonic** feel free to email me at mumpitz@wumpitz.de.

If you encounter any errors or problems with **gearthonic**, please let me know! Open an Issue at the Gitlab http://gitlab.com/wumpitz/gearthonic main repository.

C

D

G

L

M

R

S