

---

# **Gearthonic Documentation**

***Release 0.2.0***

**Timo Steidle**

August 08, 2016



<b>1 Quickstart</b>	<b>3</b>
<b>2 Contents:</b>	<b>5</b>
2.1 Usage . . . . .	5
2.2 API . . . . .	6
2.3 Contributing . . . . .	10
2.4 Changelog . . . . .	12
<b>3 Feedback</b>	<b>13</b>
<b>Python Module Index</b>	<b>15</b>



A simple client for the API of Homegear.

Look at the [documentation](#) for detailed information.



---

## Quickstart

---

Install *gearthonic* via pip:

```
pip install gearthonic
```

Initialise the client:

```
from gearthonic import GearClient
# You only have to provide the host and port of the Homegear server
gc = GearClient('192.168.1.100', 2001, secure=False, verify=False)
```

Use the predefined methods to make requests to the API:

```
gc.system.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')
```

The default communication protocol is XML-RPC. If you want to use another protocol like JSON-RPC or a MQTT broker, see the full [documentation](#).





---

**Contents:**

---

## 2.1 Usage

### 2.1.1 Basic 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', 2003)
# Now you can make your requests
gc.system.list_methods()
gc.device.get_value(1, 4, 'ACTUAL_TEMPERATURE')
```

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 Homegear API can be found in the [wiki of the Homegear project](#).

### 2.1.2 Communication protocols

Gearthonic supports different communication protocols to communicate with your Homegear server. Set the protocol while initialising the client:

```
from gearthonic import JSONRPC
gc = GearClient('192.168.1.100', 2003, protocol=JSONRPC)
```

The default protocol is XMLRPC.

Any protocol can accept additional parameters. You can supply them while initialising the client:

```
gc = GearClient('192.168.1.100', 2003, secure=False, verify=False)
```

For a complete list of available protocols and supported parameters look at [gearthonic.protocols](#).

## 2.1.3 Security

### XML RPC and JSON RPC

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.

If you are using a self signed certificate, you can skip the verification of the certificate. Set `verify=False` while initialising the `GearClient` and the certificate won't be verified. But keep in mind: that's not suggested!

### 2.1.4 Additional information

- [Documentation of all available data endpoints for all devices](#)
- [wiki of the Homegear project](#)

## 2.2 API

### 2.2.1 API Reference

See [Method Reference](#) for all API methods.

#### GearClient

Client for the APIs provided by Homegear.

**class** `gearthonic.client.GearClient` (*host, port, protocol=0, \*\*kwargs*)  
Client for the APIs provided by Homegear.

Usage:

```
>>> gc = GearClient('localhost', 1234)
>>> gc.device.list_devices()
```

The default communication protocol is XML RPC. Additionally, JSON RPC is supported:

```
>>> from gearthonic import JSONRPC
>>> GearClient('localhost', 1234, protocol=JSONRPC)
```

Any protocol can accept additional parameters. Provide them while initialising the client:

```
>>> GearClient('localhost', 1234, protocol=JSONRPC, secure=False, username='ham')
```

For a full list of supported parameters, have a look at each protocol class or see the documentation for the protocols. :class:

**call** (*method\_name, \*args, \*\*kwargs*)

Call the given method using the instantiated protocol.

#### Parameters

- **method\_name** – method to be called
- **args** – arguments passed through
- **kwargs** – keyword arguments passed through

**Returns** return value of the call

**device**

Smart access to all device related API methods.

**system**

Smart access to all system related API methods.

## Communication protocols

These classes are used to communicate with the different APIs provided by the Homegear server.

**class** gearthonic.protocols.**JsonRpcProtocol**(*host, port, secure=True, verify=True, username=None, password=None*)

Communicate with Homegear via JSON RPC.

```
>>> jp = JsonRpcProtocol('host.example.com', 2003)
>>> jp.call('listDevices')
[...]
```

Set `secure=False` to use http instead off https. Set `verify=False` to skip the verification of the SSL cert.

Provide credentials via `username` and `password` if the Homegear server is secured by basic auth. It's not possible to use authentication with an insecure connection (http)!

**call**(*method\_name, \*args, \*\*kwargs*)

Call the given method using the HTTPServer.

**Parameters**

- **method\_name** – Method to be called
- **args** – Arguments passed through
- **kwargs** – Keyword arguments passed through

**Returns** Return value of the XML RPC method

**class** gearthonic.protocols.**XmlRpcProtocol**(*host, port, secure=True, verify=True, username=None, password=None*)

Communicate with Homegear via XML RPC.

```
>>> xp = XmlRpcProtocol('host.example.com', 2003)
>>> xp.call('listDevices')
[...]
```

Set `secure=False` to use http instead off https. Set `verify=False` to skip the verification of the SSL cert.

Provide credentials via `username` and `password` if the Homegear server is secured by basic auth. It's not possible to use authentication with an insecure connection (http)!

**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

`gearthonic.protocols.initialise_protocol(protocol, host, port, **kwargs)`  
Factory method to initialise a specific protocol.

**Parameters**

- **protocol** (*int*) – ID of the protocol to initialise
- **host** (*str*) – host of the server
- **port** (*int*) – port of the server
- **kwargs** – will be used to initialise the protocol

**Return type** `_ProtocolInterface`

## 2.2.2 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.

**activate\_link\_paramset** (*peer\_id*, *channel*, *remote\_peer\_id*, *remote\_channel*,  
*long\_pressed=False*)

**Returns**

**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

## 2.3 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:

### 2.3.1 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 :)

### 2.3.2 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.

### 2.3.3 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.

## **2.3.4 Tips**

To run a subset of tests:

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

## **2.4 Changelog**

### **2.4.1 0.2.0 (2016-08-08)**

- Introduced different communication protocols (XML RPC and JSON RPC)

### **2.4.2 0.1.2 (2016-07-13)**

- Fixed broken setup.py.

### **2.4.3 0.1.1 (2016-07-13)**

- Added documentation.

### **2.4.4 0.1.0 (2016-07-13)**

- Initial release to pypi.



---

### Feedback

---

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



## g

`gearthonic.client`, [6](#)  
`gearthonic.protocols`, [7](#)



## A

activate\_link\_paramset() (gearthonic.methods.DeviceMethodsCollection method), 9

## C

call() (gearthonic.client.GearClient method), 6  
call() (gearthonic.protocols.JsonRpcProtocol method), 7  
call() (gearthonic.protocols.XmlRpcProtocol method), 7

## D

device (gearthonic.client.GearClient attribute), 7  
DeviceMethodsCollection (class in gearthonic.methods), 9

## G

GearClient (class in gearthonic.client), 6  
gearthonic.client (module), 6  
gearthonic.protocols (module), 7  
get\_capabilities() (gearthonic.methods.SystemMethodsCollection method), 8  
get\_value() (gearthonic.methods.DeviceMethodsCollection method), 9

## I

initialise\_protocol() (in module gearthonic.protocols), 8

## J

JsonRpcProtocol (class in gearthonic.protocols), 7

## L

list\_devices() (gearthonic.methods.DeviceMethodsCollection method), 9  
list\_methods() (gearthonic.methods.SystemMethodsCollection method), 8

## M

method\_help() (gearthonic.methods.SystemMethodsCollection method), 8  
method\_signature() (gearthonic.methods.SystemMethodsCollection method), 8

multicall() (gearthonic.methods.SystemMethodsCollection method), 8

## S

set\_value() (gearthonic.methods.DeviceMethodsCollection method), 10  
system (gearthonic.client.GearClient attribute), 7  
SystemMethodsCollection (class in gearthonic.methods), 8

## X

XmlRpcProtocol (class in gearthonic.protocols), 7