jsonrpclib-pelix Documentation

Release 0.4.0

Thomas Calmant

Mar 11, 2019

Contents

This library implements the JSON-RPC 2.0 proposed specification in pure Python. It is designed to be as compatible with the syntax of `xmlrpclib` as possible (it extends where possible), so that projects using `xmlrpclib` could easily be modified to use JSON and experiment with the differences.

It is backwards-compatible with the 1.0 specification, and supports all of the new proposed features of 2.0, including:

- Batch submission (via the `MultiCall` class)
- Keyword arguments
- Notifications (both in a batch and *normal*)
- Class translation using the `__jsonclass__` key.

A `SimpleJSONRPCServer` class has been added. It is intended to emulate the `SimpleXMLRPCServer` from the default Python distribution.

CHAPTER 1

Installation

1.1 Requirements

It supports `cjson` and `simplejson`, and looks for the parsers in that order (searching first for `cjson`, then for the *built-in* `json` in 2.7+, and then the `simplejson` external library). One of these must be installed to use this library, although if you have a standard distribution of 2.7+, you should already have one. Keep in mind that `cjson` is supposed to be the quickest, I believe, so if you are going for full-on optimization you may want to pick it up.

1.2 Installation

You can install the latest stable version from PyPI with the following command:

```
# Global installation
pip install jsonrpclib-pelix

# Local installation
pip install --user jsonrpclib-pelix
```

Alternatively, you can install the latest development version:

```
pip install git+https://github.com/tcalmant/jsonrpclib.git
```

Finally, you can download the source from the GitHub repository at http://github.com/tcalmant/jsonrpclib and manually install it with the following commands:

```
git clone git://github.com/tcalmant/jsonrpclib.git
cd jsonrpclib
python setup.py install
```

1.3 Tests

Tests are an almost-verbatim drop from the JSON-RPC specification 2.0 page. They can be run using *unittest* or *nosetest*:

```
python -m unittest discover tests
python3 -m unittest discover tests
nosetests tests
```

CHAPTER 2

JSON-RPC Client usage

This is (obviously) taken from a console session.

```
>>> import jsonrpclib
>>> server = jsonrpclib.ServerProxy('http://localhost:8080')
>>> server.add(5,6)
11
>>> server.add(x=5, y=10)
15
>>> server._notify.add(5,6)
# No result returned...
>>> batch = jsonrpclib.MultiCall(server)
>>> batch.add(5, 6)
>>> batch.ping({'key':'value'})
>>> batch._notify.add(4, 30)
>>> results = batch()
>>> for result in results:
>>> ... print(result)
11
{'key': 'value'}
# Note that there are only two responses -- this is according to spec.

# Clean up
>>> server('close')()

# Using client history
>>> history = jsonrpclib.history.History()
>>> server = jsonrpclib.ServerProxy('http://localhost:8080', history=history)
>>> server.add(5,6)
11
>>> print(history.request)
{"id": "f682b956-c8e1-4506-9db4-29fe8bc9fcaa", "jsonrpc": "2.0",
 "method": "add", "params": [5, 6]}
>>> print(history.response)
{"id": "f682b956-c8e1-4506-9db4-29fe8bc9fcaa", "jsonrpc": "2.0",
```

(continues on next page)

(continued from previous page)

```
 "result": 11}

# Clean up
>>> server('close')()
```

If you need 1.0 functionality, there are a bunch of places you can pass that in, although the best is just to give a specific configuration to `jsonrpclib.ServerProxy`:

```
>>> import jsonrpclib
>>> jsonrpclib.config.DEFAULT.version
2.0
>>> config = jsonrpclib.config.Config(version=1.0)
>>> history = jsonrpclib.history.History()
>>> server = jsonrpclib.ServerProxy('http://localhost:8080', config=config,
                                    history=history)
>>> server.add(7, 10)
17
>>> print(history.request)
{"id": "827b2923-5b37-49a5-8b36-e73920a16d32",
 "method": "add", "params": [7, 10]}
>>> print(history.response)
{"id": "827b2923-5b37-49a5-8b36-e73920a16d32", "error": null, "result": 17}
>>> server('close')()
```

The equivalent `loads` and `dumps` functions also exist, although with minor modifications. The `dumps` arguments are almost identical, but it adds three arguments: `rpcid` for the 'id' key, `version` to specify the JSON-RPC compatibility, and `notify` if it's a request that you want to be a notification.

Additionally, the `loads` method does not return the params and method like `xmlrpclib`, but instead a.) parses for errors, raising ProtocolErrors, and b.) returns the entire structure of the request / response for manual parsing.

2.1 Unix Socket

To connect a JSON-RPC server over a Unix socket, you have to use a specific protocol: `unix+http`.

When connecting to a Unix socket in the current working directory, you can use the following syntax: `unix+http://my.socket`

When you need to give an absolute path you must use the path part of the URL, the host part will be ignored. For example, you can use this URL to indicate a Unix socket in `/var/lib/daemon.socket`: `unix+http:///var/lib/daemon.socket`

Note: Currently, only HTTP is supported over a Unix socket. If you want HTTPS support to be implemented, please create an issue on GitHub.

2.2 Additional headers

If your remote service requires custom headers in request, you can pass them as as a `headers` keyword argument, when creating the `ServerProxy`:

```
>>> import jsonrpclib
>>> server = jsonrpclib.ServerProxy("http://localhost:8080",
                                    headers={'X-Test' : 'Test'})
```

You can also put additional request headers only for certain method invocation:

```
>>> import jsonrpclib
>>> server = jsonrpclib.ServerProxy("http://localhost:8080")
>>> with server._additional_headers({'X-Test' : 'Test'}) as test_server:
...     test_server.ping(42)
...
>>> # X-Test header will be no longer sent in requests
```

Of course `_additional_headers` contexts can be nested as well.

CHAPTER 3

Asynchronous JSON-RPC Client

Warning: This feature requires Python 3.5+

Warning: Work in progress

This feature is a work in progress. This documentation might not updated as often as the source code.

An asynchronous version of the client implementation is provided by the `jsonrpclib.jsonrpc_async` module. The latter provides the `AsyncServerProxy` class, which uses an asynchronous *Transport* implementation. Currently, the only provided *Transport* is based on *aiohttp*.

The following documentation will use the aiohttp library.

Note: `aiohttp` requires Python 3.5.3+ to work.

3.1 Sample usage

This sample shows how easy it is to use the new API.

Warning: In the current state of development, the `AsyncServerProxy` uses *aiohttp* under the hood.

The next step will be to allow the developer to use a custom *Transport* implementation.

```
import asyncio

from jsonrpclib.jsonrpc_async import AsyncServerProxy
```

(continues on next page)

(continued from previous page)

```
from jsonrpclib.impl.aiohttp_impl import AiohttpTransport


async def main():
    """
    Script entry point
    """
    # As easy as it can be
    server = AsyncServerProxy("http://localhost:8080", AiohttpTransport)
    print(await server.pow(2, 4096))


if __name__ == "__main__":
    # Use an event loop to run the asynchronous entry point
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```

CHAPTER 4

Simple JSON-RPC Server

This is identical in usage (or should be) to the `SimpleXMLRPCServer` in the Python standard library. Some of the differences in features are that it obviously supports notification, batch calls, class translation (if left on), etc. Note: The import line is slightly different from the regular `SimpleXMLRPCServer`, since the `SimpleJSONRPCServer` is provided by the `jsonrpclib` library.

```
from jsonrpclib.SimpleJSONRPCServer import SimpleJSONRPCServer

server = SimpleJSONRPCServer(('localhost', 8080))
server.register_function(pow)
server.register_function(lambda x,y: x+y, 'add')
server.register_function(lambda x: x, 'ping')
server.serve_forever()
```

To start protect the server with SSL, use the following snippet:

```
from jsonrpclib.SimpleJSONRPCServer import SimpleJSONRPCServer

# Setup the SSL socket
server = SimpleJSONRPCServer(('localhost', 8080), bind_and_activate=False)
server.socket = ssl.wrap_socket(server.socket, certfile='server.pem',
                                server_side=True)
server.server_bind()
server.server_activate()

# ... register functions
# Start the server
server.serve_forever()
```

4.1 Note on performances

Sometimes, it might seen that a client is really slow connecting the server. Chances are this is due to the fact that your server is listening to IPv4 packets only, whereas clients know both your IPv6 and IPv4 addresses. In this situation,

clients wait a timeout of around 1 second for the IPv6 address to response before trying the IPv4 one.

To avoid this problem, you will have to start the server in IPv6 mode and to activate the double stack mode. That way, the server will be accessible with both IPv4 and IPv6 addresses. Note that to be sure this works, it is recommended that the server binds all IPv6 interfaces (`::`).

This can be done using the following arguments when creating the server:

```
import socket
from jsonrpclib.SimpleJSONRPCServer import SimpleJSONRPCServer

server = SimpleJSONRPCServer(
   ("::", 8080),
   address_family=socket.AF_INET6,
   use_double_stack=True
)
```

4.2 Notification Thread Pool

By default, notification calls are handled in the request handling thread. It is possible to use a thread pool to handle them, by giving it to the server using the `set_notification_pool()` method:

```
from jsonrpclib.SimpleJSONRPCServer import SimpleJSONRPCServer
from jsonrpclib.threadpool import ThreadPool

# Setup the thread pool: between 0 and 10 threads
pool = ThreadPool(max_threads=10, min_threads=0)

# Don't forget to start it
pool.start()

# Setup the server
server = SimpleJSONRPCServer(('localhost', 8080), config)
server.set_notification_pool(pool)

# Register methods
server.register_function(pow)
server.register_function(lambda x,y: x+y, 'add')
server.register_function(lambda x: x, 'ping')

try:
    server.serve_forever()
finally:
    # Stop the thread pool (let threads finish their current task)
    pool.stop()
    server.set_notification_pool(None)
```

4.3 Threaded server

It is also possible to use a thread pool to handle clients requests, using the `PooledJSONRPCServer` class. By default, this class uses pool of 0 to 30 threads. A custom pool can be given with the `thread_pool` parameter of the class constructor.

The notification pool and the request pool are different: by default, a server with a request pool doesn't have a notification pool.

```
from jsonrpclib.SimpleJSONRPCServer import PooledJSONRPCServer
from jsonrpclib.threadpool import ThreadPool

# Setup the notification and request pools
nofif_pool = ThreadPool(max_threads=10, min_threads=0)
request_pool = ThreadPool(max_threads=50, min_threads=10)

# Don't forget to start them
nofif_pool.start()
request_pool.start()

# Setup the server
server = PooledJSONRPCServer(('localhost', 8080), config,
                             thread_pool=request_pool)
server.set_notification_pool(nofif_pool)

# Register methods
server.register_function(pow)
server.register_function(lambda x,y: x+y, 'add')
server.register_function(lambda x: x, 'ping')

try:
    server.serve_forever()
finally:
    # Stop the thread pools (let threads finish their current task)
    request_pool.stop()
    nofif_pool.stop()
    server.set_notification_pool(None)
```

4.4 Unix Socket

To start a server listening on a Unix socket, you will have to use the following snippet:

```
from jsonrpclib.SimpleJSONRPCServer import SimpleJSONRPCServer
import os
import socket

# Set the path to the socket file
socket_name = "/tmp/my_socket.socket"

# Ensure that the file doesn't exist yet (or an error will be raised)
if os.path.exists(socket_name):
   os.remove(socket_name)

try:
   # Start the server, indicating the socket family
   # The server will force some flags when in Unix socket mode
   # (no log request, no reuse address, ...)
   srv = SimpleJSONRPCServer(socket_name, address_family=socket.AF_UNIX)

   # ... register methods to the server
   # Run the server
```

(continues on next page)

(continued from previous page)

```
    srv.serve_forever()
except KeyboardInterrupt:
    # Shutdown the server gracefully
    srv.shutdown()
    srv.server_close()
finally:
    # You should clean up after the server stopped
    os.remove(socket_name)
```

This feature is tested on Linux during Travis-CI builds. It also has been tested on Windows Subsystem for Linux (WSL) on Windows 10 1809.

This feature is not available on "pure" Windows, as it doesn't provide the `AF_UNIX` address family.

CHAPTER 5

Asynchronous JSON-RPC Server

Warning: This feature requires Python 3.5+

An asynchronous version of the server protocol is provided by the `jsonrpclib.server_protocol_async` module. The latter provides the `AsyncJsonRpcProtocolHandler` class, which can be used in any `asyncio` protocol implementation. Currently, the library comes with a server implementation based on the aiohttp library.

Other implementations can be implemented/contributed.

5.1 Sample usage with the `aiohttp` implementation

5.1.1 Imports

The `aiohttp` module is not explicitly imported as the `AiohttpJsonRpcServer` class hides all the initialization process. If you want to use a custom `aiohttp` instance, you can register the low-level request handler: `AiohttpRequestHandler`.

A high level API request handler will be implemented for version 0.5.0.

```
import asyncio
from jsonrpclib.server_protocol_async import AsyncJsonRpcProtocolHandler
from jsonrpclib.impl.aiohttp_impl import AiohttpRequestHandler, AiohttpJsonRpcServer
```

5.1.2 Prepare the protocol handler

The first step is the creation of the protocol handler. It has the same API as the simple JSON-RPC/XML-RPC servers to register functions.

```
async def my_async_method():
    # Do something...

def my_sync_method():
    # Do something...

# Prepare the protocol handler
json_handler = AsyncJsonRpcProtocolHandler()

# Register functions the same way
json_handler.register_function(my_async_method)
json_handler.register_function(my_sync_method)

# Lambda still works
json_handler.register_function(lambda: "Hello", name="hello")

# As well as introspection methods
json_handler.register_introspection_functions()
```

5.1.3 Asynchronous start method

We then define a utility method that will start the `aiohttp` server in the current event loop. It will also start a *checker* which will wake up every half second to ensure that Python looks for `KeyboardInterrupt` exceptions to raise from time to time:

```
def start_sync(srv):
    loop = asyncio.get_event_loop()
    checker = loop.create_task(srv.async_check_interrupt())
    try:
        loop.run_until_complete(srv.run())
    except KeyboardInterrupt:
        srv.shutdown()
    finally:
        # Wait for the interruption checker
        loop.run_until_complete(checker)
```

5.1.4 Execution

Here, we can manage the life cycle of the HTTP server.

We first create the HTTP request handler based on `aiohttp`. It is a low-level request handler, which is why it's there that we indicate the path used for JSON-RPC queries.

Then, we prepare the `aiohttp`-based server itself, indicating its request handler, binding address and listened port:

```
http_handler = AiohttpRequestHandler(json_handler, "/json-rpc")
srv = AiohttpJsonRpcServer(http_handler, "localhost", 8080)
try:
    start_sync()
except KeyboardInterrupt:
    srv.shutdown()
```

The endpoint is now accessible on http://localhost:8080/json-rpc.

5.2 Implement a new asynchronous transport

Warning: TODO

1. Inherit `AbstractAsyncTransport`
2. Implement `request(self, host, handler, request_body, verbose=False)`

CHAPTER 6

Class Translation

The library supports an *"automatic"* class translation process, although it is turned off by default. This can be devastatingly slow if improperly used, so the following is just a short list of things to keep in mind when using it.

- Keep It (the object) Simple Stupid. (for exceptions, keep reading)
- Do not require init params (for exceptions, keep reading)
- Getter properties without setters could be dangerous (read: not tested)

If any of the above are issues, use the `_serialize` method. (see usage below) The server and client must **BOTH** have `use_jsonclass` configuration item on and they must both have access to the same libraries used by the objects for this to work.

If you have excessively nested arguments, it would be better to turn off the translation and manually invoke it on specific objects using `jsonrpclib.jsonclass.dump` / `jsonrpclib.jsonclass.load` (since the default behavior recursively goes through attributes and lists / dicts / tuples).

- Sample file: `test_obj.py`

```
# This object is /very/ simple, and the system will look through the
# attributes and serialize what it can.
class TestObj(object):
    foo = 'bar'

# This object requires __init__ params, so it uses the _serialize method
# and returns a tuple of init params and attribute values (the init params
# can be a dict or a list, but the attribute values must be a dict.)
class TestSerial(object):
    foo = 'bar'
    def __init__(self, *args):
        self.args = args
    def _serialize(self):
        return (self.args, {'foo':self.foo,})
```

- Sample usage:

```
>>> import jsonrpclib
>>> import test_obj

# History is used only to print the serialized form of beans
>>> history = jsonrpclib.history.History()
>>> testobj1 = test_obj.TestObj()
>>> testobj2 = test_obj.TestSerial()
>>> server = jsonrpclib.Server('http://localhost:8080', history=history)

# The 'ping' just returns whatever is sent
>>> ping1 = server.ping(testobj1)
>>> ping2 = server.ping(testobj2)

>>> print(history.request)
{"id": "7805f1f9-9abd-49c6-81dc-dbd47229fe13", "jsonrpc": "2.0",
 "method": "ping", "params": [{"__jsonclass__":
                               ["test_obj.TestSerial", []], "foo": "bar"}
                             ]}
>>> print(history.response)
{"id": "7805f1f9-9abd-49c6-81dc-dbd47229fe13", "jsonrpc": "2.0",
 "result": {"__jsonclass__": ["test_obj.TestSerial", []], "foo": "bar"}}
```

This behavior is turned on by default. To deactivate it, just set the `use_jsonclass` member of a server `Config` to `False`. If you want to use a per-class serialization method, set its name in the `serialize_method` member of a server `Config`. Finally, if you are using classes that you have defined in the implementation (as in, not a separate library), you'll need to add those (on **BOTH** the server and the client) using the `config.classes.add()` method.

Feedback on this "feature" is very, VERY much appreciated.

CHAPTER 7

Release Notes

7.1 0.4.0

Release Date 2019-01-13

- Added back support of Unix sockets on both server and client side. **Note:** HTTPS is not supported on server-side Unix sockets
- Fixed the CGI request handler
- Fixed the request handler wrapping on server side
- Documentation is now hosted on ReadTheDocs: https://jsonrpclib-pelix.readthedocs.io/

7.2 0.3.2

Release Date 2018-10-26

- Fixed a memory leak in the Thread Pool, causing the `PooledJSONRPCServer` to crash after some uptime (see #35). Thanks @animalmutch for reporting it.

7.3 0.3.1

Release Date 2017-06-27

- Hide *dunder* methods from remote calls (thanks to @MarcSchmitzer). This avoids weird behaviours with special/meta methods (`__len__`, `__add__`, ...). See (#32) for reference.

7.4 0.3.0

Release Date 2017-04-27

- Handle the potentially incomplete `xmlrpc.server` package when the future package is used (thanks to @MarcSchmitzer)

7.5 0.2.9

Release Date 2016-12-12

- Added support for enumerations (`enum.Enum` classes, added in Python 3.4)
- Removed tests for `pypy3` as it doesn't work with `pip` anymore

7.6 0.2.8

Release Date 2016-08-23

- Clients can now connect servers using basic authentication. The server URL must be given using this format: http://user:password@server
- The thread pool has been updated to reflect the fixes contributed by @Paltoquet for the iPOPO project.

7.7 0.2.7

Release Date 2016-06-12

- Application of the `TransportMixin` fix developped by @MarcSchmitzer (#26).

7.8 0.2.6

Release Date 2015-08-24

- Removed support for Python 2.6
- Added a `__repr__` method to the `_Method` class
- Project is now tested against Python 3.4 and Pypy 3 on Travis-CI

7.9 0.2.5

Release Date 2015-02-28

- Corrects the `PooledJSONRPCServer`
- Stops the thread pool of the `PooledJSONRPCServer` in `server_close()`
- Corrects the `Config.copy()` method: it now uses a copy of local classes and serialization handlers instead of sharing those dictionaries.

7.10 0.2.4

Release Date 2015-02-16

- Added a thread pool to handle requests
- Corrects the handling of reused request sockets on the server side
- Corrects the `additional_header` feature: now supports different headers for different proxies, from @MarcSchmitzer
- Adds a `data` field to error responses, from @MarcSchmitzer and @mbra

7.11 0.2.3

Release Date 2015-01-16

- Added support for a custom `SSLContext` on client side

7.12 0.2.2

Release Date 2014-12-23

- Fixed support for IronPython
- Fixed Python 2.6 compatibility in tests
- Added logs on server side

7.13 0.2.1

Release Date 2014-09-18

- Return `None` instead of an empty list on empty replies
- Better lookup of the custom serializer to look for

7.14 0.2.0

Release Date 2014-08-28

- Code review
- Fixed propagation of configuration through `jsonclass`, from dawryn

7.15 0.1.9

Release Date 2014-06-09

- Fixed compatibility with JSON-RPC 1.0
- Propagate configuration through `jsonclass`, from dawryn

7.16 0.1.8

Release Date 2014-06-05

- Enhanced support for bean inheritance

7.17 0.1.7

Release Date 2014-06-02

- Enhanced support of custom objects (with `__slots__` and handlers), from dawryn See Pull requests #5, #6, #7)
- Added tests
- First upload as a Wheel file

7.18 0.1.6.1

Release Date 2013-10-25

- Fixed loading of recursive bean fields (beans can contain other beans)
- `ServerProxy` can now be closed using: `client("close")()`

7.19 0.1.6

Release Date 2013-10-14

- Fixed bean marshalling
- Added support for `set` and `frozenset` values
- Changed configuration singleton to `Config` instances

7.20 0.1.5

Release Date 2013-06-20

- Requests with ID 0 are not considered notifications anymore
- Fixed memory leak due to keeping history in `ServerProxy`
- `Content-Type` can be configured
- Better feeding of the JSON parser (avoid missing parts of a multi-bytes character)
- Code formatting/compatibility enhancements
- Applied enhancements found on other forks:
  - Less strict error response handling from drdaeman
  - In case of a non-predefined error, raise an `AppError` and give access to *error.data*, from tuomassalo

7.21 0.1.4

Release Date 2013-05-22

- First published version of this fork, with support for Python 3
- Version number was following the original project one

CHAPTER 8

License

iPOPO is licensed under the terms of the Apache Software License 2.0. All contributions must comply with this license.

8.1 File Header

This snippet is added to the module-level documentation:

```
Copyright 2019 Thomas Calmant

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

8.2 License Full Text

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
```

(continues on next page)

(continued from previous page)

```
1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
```

(continues on next page)

(continued from previous page)

```
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
```

(continues on next page)

(continued from previous page)

```
        that such additional attribution notices cannot be construed
        as modifying the License.

    You may add Your own copyright statement to Your modifications and
    may provide additional or different license terms and conditions
    for use, reproduction, or distribution of Your modifications, or
    for any such Derivative Works as a whole, provided Your use,
    reproduction, and distribution of the Work otherwise complies with
    the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
    any Contribution intentionally submitted for inclusion in the Work
    by You to the Licensor shall be under the terms and conditions of
    this License, without any additional terms or conditions.
    Notwithstanding the above, nothing herein shall supersede or modify
    the terms of any separate license agreement you may have executed
    with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
    names, trademarks, service marks, or product names of the Licensor,
    except as required for reasonable and customary use in describing the
    origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
    agreed to in writing, Licensor provides the Work (and each
    Contributor provides its Contributions) on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied, including, without limitation, any warranties or conditions
    of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
    PARTICULAR PURPOSE. You are solely responsible for determining the
    appropriateness of using or redistributing the Work and assume any
    risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
    whether in tort (including negligence), contract, or otherwise,
    unless required by applicable law (such as deliberate and grossly
    negligent acts) or agreed to in writing, shall any Contributor be
    liable to You for damages, including any direct, indirect, special,
    incidental, or consequential damages of any character arising as a
    result of this License or out of the use or inability to use the
    Work (including but not limited to damages for loss of goodwill,
    work stoppage, computer failure or malfunction, or any and all
    other commercial damages or losses), even if such Contributor
    has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
    the Work or Derivative Works thereof, You may choose to offer,
    and charge a fee for, acceptance of support, warranty, indemnity,
    or other liability obligations and/or rights consistent with this
    License. However, in accepting such obligations, You may act only
    on Your own behalf and on Your sole responsibility, not on behalf
    of any other Contributor, and only if You agree to indemnify,
    defend, and hold each Contributor harmless for any liability
    incurred by, or claims asserted against, such Contributor by reason
    of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
```

(continues on next page)

(continued from previous page)

```
APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

CHAPTER 9

Why JSON-RPC?

In my opinion, there are several reasons to choose JSON over XML for RPC:

- Much simpler to read (I suppose this is opinion, but I know I'm right. :)
- Size / Bandwidth - Main reason, a JSON object representation is just much smaller.
- Parsing - JSON should be much quicker to parse than XML.
- Easy class passing with `jsonclass` (when enabled)

In the interest of being fair, there are also a few reasons to choose XML over JSON:

- Your server doesn't do JSON (rather obvious)
- Wider XML-RPC support across APIs (can we change this? :))
- Libraries are more established, i.e. more stable (Let's change this too.)

CHAPTER 10

About this version

This is a patched version of the original `jsonrpclib` project by Josh Marshall, available at https://github.com/joshmarshall/jsonrpclib.

The suffix *-pelix* only indicates that this version works with Pelix Remote Services, but it is **not** a Pelix specific implementation.

- This version adds support for Python 3, staying compatible with Python 2.7.
- It is now possible to use the dispatch_method argument while extending the SimpleJSONRPCDispatcher, to use a custom dispatcher. This allows to use this package by Pelix Remote Services.
- It can use thread pools to control the number of threads spawned to handle notification requests and clients connections.
- The modifications added in other forks of this project have been added:
  - From https://github.com/drdaeman/jsonrpclib:
    - Improved JSON-RPC 1.0 support
    - Less strict error response handling
  - From https://github.com/tuomassalo/jsonrpclib:
    - In case of a non-pre-defined error, raise an AppError and give access to *error.data*
  - From https://github.com/dejw/jsonrpclib:
    - Custom headers can be sent with request and associated tests
- Since version 0.4, this package added back the support of Unix sockets.
- This version cannot be installed with the original `jsonrpclib`, as it uses the same package name.