Python EPC Documentation

Release 0.0.5

Takafumi Arakaki

Nov 03, 2018

Contents

Links:

- Documentation (at Read the Docs)
- Repository (at GitHub)
- Issue tracker (at GitHub)
- PyPI
- Travis CI

Other resources:

- kiwanami/emacs-epc (Client and server implementation in Emacs Lisp and Perl.)
- tkf/emacs-jedi (Python completion for Emacs using EPC server.)

CHAPTER 1

What is this?

EPC is an RPC stack for Emacs Lisp and Python-EPC is its server side and client side implementation in Python. Using Python-EPC, you can easily call Emacs Lisp functions from Python and Python functions from Emacs. For example, you can use Python GUI module to build widgets for Emacs (see examples/gtk/server.py for example).

Python-EPC is tested against Python 2.6, 2.7, 3.2 and 3.3.

CHAPTER 2

Install

To install Python-EPC and its dependency sexpdata, run the following command.:

```
pip install epc
```

CHAPTER 3

Usage

Save the following code as `my-server.py`. (You can find functionally the same code in examples/echo/server.py):

```
from epc.server import EPCServer

server = EPCServer(('localhost', 0))

@server.register_function
def echo(*a):
    return a

server.print_port()
server.serve_forever()
```

And then run the following code from Emacs. This is a stripped version of examples/echo/client.el included in Python-EPC repository.:

```
(require 'epc)

(defvar my-epc (epc:start-epc "python" '("my-server.py")))

(deferred:$
  (epc:call-deferred my-epc 'echo '(10))
  (deferred:nextc it
    (lambda (x) (message "Return : %S" x))))

(message "Return : %S" (epc:call-sync my-epc 'echo '(10 40)))
```

If you have carton installed, you can run the above sample by simply typing the following commands:

```
make elpa         # install EPC in a separated environment
make run-sample   # run examples/echo/client.el
```

For example of bidirectional communication and integration with GTK, see examples/gtk/server.py. You can run this example by:

```
make elpa
make run-gtk-sample  # run examples/gtk/client.el
```

CHAPTER 4

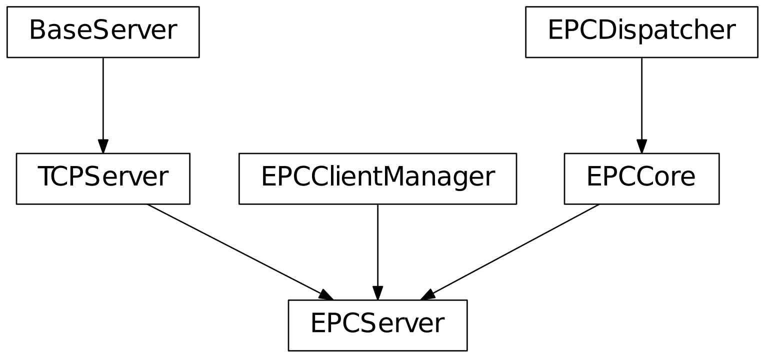
License

Python-EPC is licensed under GPL v3. See COPYING for details.

CHAPTER 5

EPC server API

5.1 Server



class epc.server.**EPCServer**(*server_address*, *RequestHandlerClass=<class epc.handler.EPCHandler>*, *bind_and_activate=True*, *debugger=None*, *log_traceback=False*)

A server class to publish functions and call functions via EPC protocol.

To publish Python functions, all you need is `register_function()`, `print_port()` and `serve_forever()`.

```
>>> server = EPCServer(('localhost', 0))
>>> def echo(*a):
...     return a
```

(continues on next page)

(continued from previous page)

```
>>> server.register_function(echo)                   #doctest: +ELLIPSIS
<function echo at 0x...>
>>> server.print_port()                                  #doctest: +SKIP
9999
>>> server.serve_forever()                               #doctest: +SKIP
```

To call client's method, use `clients` attribute to get client handler and use its *`EPCHandler.call()`* and *`EPCHandler.methods()`* methods to communicate with connected client.

```
>>> handler = server.clients[0]                          #doctest: +SKIP
>>> def callback(reply):
...     print(reply)
>>> handler.call('method_name', ['arg-1', 'arg-2', 'arg-3'],
...                 callback)                            #doctest: +SKIP
```

See `SocketServer.TCPServer` and `SocketServer.BaseServer` for other usable methods.

`register_function`(*function*, *name=None*)
Register function to be called from EPC client.

Parameters

- **`function`** (*`callable`*) – Function to publish.
- **`name`** (*`str`*) – Name by which function is published.

This method returns the given *function* as-is, so that you can use it as a decorator.

`register_instance`(*instance*, *allow_dotted_names=False*)
Register an instance to respond to EPC requests.

Parameters

- **`instance`** (*`object`*) – An object with methods to provide to peer. If this instance has *_get_method* method, EPC method name resolution can be done by this method.
- **`allow_dotted_names`** (*`bool`*) – If it is true, method names containing dots are supported. They are resolved using *getattr* for each part of the name as long as it does not start with '_'.

Unlike *`register_function()`*, only one instance can be registered.

`set_debugger`(*debugger*)
Set debugger to run when an error occurs in published method.

You can also set debugger by passing *debugger* argument to the class constructor.

Parameters **`debugger`** (*`{'pdb', 'ipdb', None}`*) – type of debugger.

`print_port`(*stream=<open file '<stdout>', mode 'w'>*)
Print port this EPC server runs on.

As Emacs client reads port number from STDOUT, you need to call this just before calling `serve_forever()`.

Parameters **`stream`** (*`text stream`*) – A stream object to write port on. Default is `sys.stdout`.

`clients = []`
A list of *`EPCHandler`* object for connected clients.

`handle_client_connect`(*handler*)
Handler which is called with a newly connected *client*.

Parameters **`handler`** (*`EPCHandler`*) – Object for handling request from the client.

Default implementation does nothing.

`handle_client_disconnect`(*handler*)
Handler which is called with a disconnected *client*.

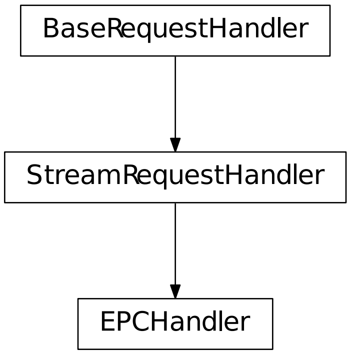
Parameters **`handler`** (*`EPCHandler`*) – Object for handling request from the client.

Default implementation does nothing.

`class` `epc.server.`**`ThreadingEPCServer`**(**args*, ***kwds*)
Class *`EPCServer`* mixed with `SocketServer.ThreadingMixIn`.

Use this class when combining EPCServer with other Python module which has event loop, such as GUI modules. For example, see examples/gtk/server.py for how to use this class with GTK

5.2 Handler



`class` `epc.server.`**`EPCHandler`**(*request*, *client_address*, *server*)

`handle_error`(*err*)
Handle error which is not handled by errback.

Parameters **`err`** (*`Exception`*) – An error not handled by other mechanisms.

Return type boolean

Return True from this function means that error is properly handled, so the error is not sent to client. Do not confuse this with `SocketServer.BaseServer.handle_error()`. This method is for handling error for each client, not for entire server. Default implementation logs the error and returns True if the error is coming from remote[1] or returns False otherwise. Therefore, only the error occurs in this handler class is sent to remote.

`call`(*name*, **args*, ***kwds*)
Call method connected to this handler.

[1] More specifically, it returns True if *err* is an instance of `BaseRemoteError` or `EPCClosed`.

Parameters

- **`name`** (*`str`*) – Method name to call.
- **`args`** (*`list`*) – Arguments for remote method to call.
- **`callback`** (*`callable`*) – A function to be called with returned value of the remote method.
- **`errback`** (*`callable`*) – A function to be called with an error occurred in the remote method. It is either an instance of `ReturnError` or `EPCError`.

`methods`(**args*, ***kwds*)

Request info of callable remote methods.

Arguments for `call()` except for *name* can be applied to this function too.

`call_sync`(*name*, *args*, *timeout=None*)

Blocking version of `call()`.

Parameters

- **`name`** (*`str`*) – Remote function name to call.
- **`args`** (*`list`*) – Arguments passed to the remote function.
- **`timeout`** (*`int or None`*) – Timeout in second. None means no timeout.

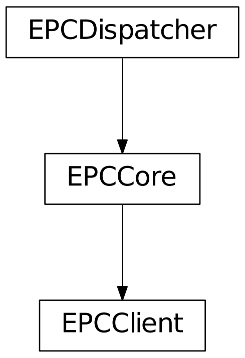
If the called remote function raise an exception, this method raise an exception. If you give *timeout*, this method may raise an *Empty* exception.

`methods_sync`(*timeout=None*)

Blocking version of `methods()`. See also `call_sync()`.

CHAPTER 6

EPC client API



class epc.client.**EPCClient**(*socket_or_address=None, debugger=None, log_traceback=False*)

EPC client class to call remote functions and serve Python functions.

```
>>> client = EPCClient()
>>> client.connect(('localhost', 9999))                  #doctest: +SKIP
>>> client.call_sync('echo', [111, 222, 333])            #doctest: +SKIP
[111, 222, 333]
```

To serve Python functions, you can use `register_function()`.

```
>>> client.register_function(str.upper)
<method 'upper' of 'str' objects>
```

`register_function()` can be used as a decorator.

```
>>> @client.register_function
... def add(x, y):
...     return x + y
```

Also, you can initialize client and connect to the server by one line.

```
>>> client = EPCClient(('localhost', 9999))              #doctest: +SKIP
```

call()
    Alias of *epc.server.EPCHandler.call()*.

call_sync()
    Alias of *epc.server.EPCHandler.call_sync()*.

methods()
    Alias of *epc.server.EPCHandler.methods()*.

methods_sync()
    Alias of *epc.server.EPCHandler.methods_sync()*.

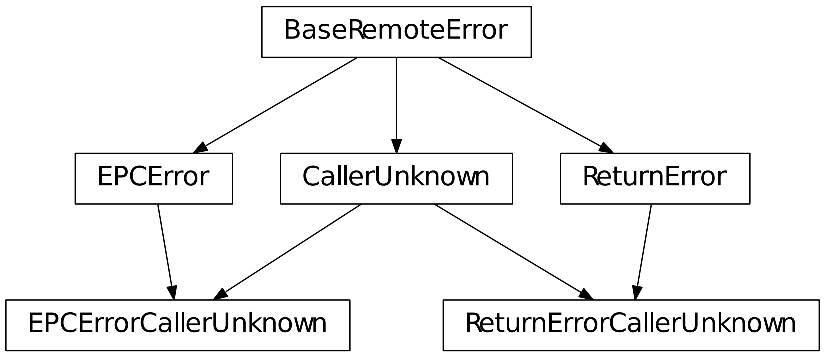
connect(*socket_or_address*)
    Connect to server and start serving registered functions.

Parameters **socket_or_address** (*tuple or socket object*) – A (host, port) pair to be passed to *socket.create_connection*, or a socket object.

close()
    Close connection.

CHAPTER 7

EPC exceptions



class epc.handler.**BaseRemoteError**
All exceptions from remote method are derived from this class.

class epc.handler.**CallerUnknown**
Error raised in remote method, but caller of the method is unknown.

class epc.handler.**EPCError**
Error returned by *epc-error* protocol.

class epc.handler.**ReturnError**
Error returned by *return-error* protocol.

class epc.handler.**EPCErrorCallerUnknown**
Same as *EPCError*, but caller is unknown.

class epc.handler.**ReturnErrorCallerUnknown**
Same as *ReturnError*, but caller is unknown.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index