The `SimpleXMLRPCServer` module provides a basic server framework for XML-RPC servers written in Python. Servers can either be free standing, using `SimpleXMLRPCServer`, or embedded in a CGI environment, using `CGIXMLRPCRequestHandler`.

```python
class SimpleXMLRPCServer.SimpleXMLRPCServer(addr, requestHandler, logRequests, allow_none, encoding, bind_and_activate)
```

Create a new server instance. This class provides methods for registration of functions that can be called by the XML-RPC protocol. The `requestHandler` parameter should be a factory for request handler instances; it defaults to `CGIXMLRPCRequestHandler`. The `addr` and `requestHandler` parameters are passed to the `SocketServer.TCPServer` constructor. If `logRequests` is true (the default), requests will be logged; setting this parameter to false will turn off logging. The `allow_none` and `encoding` parameters are passed on to `xmlrpclib` and control the XML-RPC responses that will be returned from the server. The `bind_and_activate` parameter controls whether `server_bind()` and `server_activate()` are called immediately by the constructor; it defaults to true. Setting it to false allows code to manipulate the `allow_reuse_address` class variable before the address is bound.

*Changed in version 2.5:* The `allow_none` and `encoding` parameters were added.

*Changed in version 2.6:* The `bind_and_activate` parameter was added.

```python
class SimpleXMLRPCServer.CGIXMLRPCRequestHandler(allow_none, encoding)
```

Create a new instance to handle XML-RPC requests in a CGI environment. The `allow_none` and `encoding` parameters are passed on to `xmlrpclib` and control the XML-RPC responses that will be returned from the server.

*New in version 2.3.*

*Changed in version 2.5:* The `allow_none` and `encoding` parameters were added.

```python
class SimpleXMLRPCServer.SimpleXMLRPCRequestHandler
```

Create a new request handler instance. This request handler supports POST requests and modifies logging so that the `logRequests` parameter to the `SimpleXMLRPCServer` constructor parameter is honored.

### 20.24.1. SimpleXMLRPCServer Objects

The `SimpleXMLRPCServer` class is based on `SocketServer.TCPServer` and provides a means of creating simple, stand alone XML-RPC servers.
```
SimpleXMLRPCServer.register_function(function[, name])

Register a function that can respond to XML-RPC requests. If name is given, it will be the method name associated with function, otherwise function.__name__ will be used. name can be either a normal or Unicode string, and may contain characters not legal in Python identifiers, including the period character.

SimpleXMLRPCServer.register_instance(instance[, allow_dotted_names])

Register an object which is used to expose method names which have not been registered using register_function(). If instance contains a _dispatch() method, it is called with the requested method name and the parameters from the request. Its API is def _dispatch(self, method, params) (note that params does not represent a variable argument list). If it calls an underlying function to perform its task, that function is called as func(*params), expanding the parameter list. The return value from _dispatch() is returned to the client as the result. If instance does not have a _dispatch() method, it is searched for an attribute matching the name of the requested method.

If the optional allow_dotted_names argument is true and the instance does not have a _dispatch() method, then if the requested method name contains periods, each component of the method name is searched for individually, with the effect that a simple hierarchical search is performed. The value found from this search is then called with the parameters from the request, and the return value is passed back to the client.

Warning: Enabling the allow_dotted_names option allows intruders to access your module’s global variables and may allow intruders to execute arbitrary code on your machine. Only use this option on a secure, closed network.

Changed in version 2.3.5: 2.4.1 allow_dotted_names was added to plug a security hole; prior versions are insecure.

SimpleXMLRPCServer.register_introspection_functions()

Registers the XML-RPC introspection functions system.listMethods, system.methodHelp and system.methodSignature.

New in version 2.3.

SimpleXMLRPCServer.register_multicall_functions()

Registers the XML-RPC multicall function system.multicall.

SimpleXMLRPCRequestHandler.rpc_paths

An attribute value that must be a tuple listing valid path portions of the URL for receiving XML-RPC requests. Requests posted to other paths will result in a 404 “no such page” HTTP error. If this tuple is empty, all paths will be considered valid. The default value is ('/', '/RPC2').

New in version 2.5.

SimpleXMLRPCRequestHandler.encode_threshold
```
If this attribute is not None, responses larger than this value will be encoded using the gzip transfer encoding, if permitted by the client. The default is 1400 which corresponds roughly to a single TCP packet.

New in version 2.7.

20.24.1. SimpleXMLRPCServer Example

Server code:

```python
from SimpleXMLRPCServer import SimpleXMLRPCServer
from SimpleXMLRPCServer import SimpleXMLRPCRequestHandler

# Restrict to a particular path.
class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)

# Create server
server = SimpleXMLRPCServer(('localhost', 8000),
                             requestHandler=RequestHandler)
server.register_introspection_functions()

# Register pow() function; this will use the value of
# pow.__name__ as the name, which is just 'pow'.
server.register_function(pow)

# Register a function under a different name
def adder_function(x, y):
    return x + y
server.register_function(adder_function, 'add')

# Register an instance; all the methods of the instance are
# published as XML-RPC methods (in this case, just 'div').
class MyFuncs:
    def div(self, x, y):
        return x // y
server.register_instance(MyFuncs())

# Run the server's main loop
server.serve_forever()
```

The following client code will call the methods made available by the preceding server:

```python
import xmlrpclib

s = xmlrpclib.ServerProxy('http://localhost:8000')
print s.pow(2, 3)  # Returns 2**3 = 8
print s.add(2, 3)  # Returns 5
print s.div(5, 2)  # Returns 5//2 = 2
```
20.24.2. CGIXMLRPCRequestHandler

The CGIXMLRPCRequestHandler class can be used to handle XML-RPC requests sent to Python CGI scripts.

CGIXMLRPCRequestHandler.register_function(function[, name])

Register a function that can respond to XML-RPC requests. If name is given, it will be the method name associated with function, otherwise function.__name__ will be used. name can be either a normal or Unicode string, and may contain characters not legal in Python identifiers, including the period character.

CGIXMLRPCRequestHandler.register_instance(instance)

Register an object which is used to expose method names which have not been registered using register_function(). If instance contains a _dispatch() method, it is called with the requested method name and the parameters from the request; the return value is returned to the client as the result. If instance does not have a _dispatch() method, it is searched for an attribute matching the name of the requested method; if the requested method name contains periods, each component of the method name is searched for individually, with the effect that a simple hierarchical search is performed. The value found from this search is then called with the parameters from the request, and the return value is passed back to the client.

CGIXMLRPCRequestHandler.register_introspection_functions()

Register the XML-RPC introspection functions system.listMethods, system.methodHelp and system.methodSignature.

CGIXMLRPCRequestHandler.register_multicall_functions()

Register the XML-RPC multicall function system.multicall.

CGIXMLRPCRequestHandler.handle_request([request_text = None])

Handle a XML-RPC request. If request_text is given, it should be the POST data provided by the HTTP server, otherwise the contents of stdin will be used.

Example:

class MyFuncs:
    def div(self, x, y) : return x // y

handler = CGIXMLRPCRequestHandler()
handler.register_function(pow)
handler.register_function(lambda x,y: x+y, 'add')
handler.register_introspection_functions()
handler.register_instance(MyFuncs())
handler.handle_request()