CHAPTER 3

RNA Over XML-RPC

This chapter describes the Remote Node Abstraction (RNA) subservice and includes the following sections:

Overview

The RNA subservice enables you to easily interact with Mediator framework nodes. Using RNA, you can query and manipulate nodes, retrieve and override values, and so on.The combination of the built-in library, its verbose mode output, and the Python interactive prompt, make RNA a very good tool to use for testing and troubleshooting XML-RPC interactions. If you are developing RNA over XML-RPC-based application code, we recommend that you use Python to assist with development.

RNA over XML-RPCv2 uses the request path or URL to specify the URL of the node against which an XML-RPCv2 method invocation should be made. Using the Python built-in libraries means creating an instance of the XML-RPC server proxy for each node you wish to interact with.

Creating a Node Proxy

A proxy is a client-side object representation of a remote resource or object (a framework node or service, in this case). Proxies are a common tool used to simplify client-side RPC usage. Not all XML-RPC libraries use proxies, but proxy-based XML-RPC support is available for most environments.

This section describes how to create a node proxy with the help of the following example:

```
>>> import xmlrpclib
>>> relay =
xmlrpclib.ServerProxy("https://mpxadmin:mpxadmin@beggar/XMLRPCv2/RNA/interfaces/relay1",
verbose=True)
```

The server proxy is directly associated with the framework node and each framework node being interacted with will require its own server proxy instance.The credentials have been provided using the URL syntax. The Python XML-RPC library supports the use of this syntax to specify the contents of a BASIC authorization header. It is only because the library supports this syntax that we are able to use it

in the manner. Other libraries may support BASIC authorization in different manners, or perhaps not at all. For libraries that do not support it at all, a proper BASIC authorization header the XML-RPC requests explicitly must be added or attached.

Using a Node Proxy

This section describes how to use a node proxy with the help of APIs and includes the following topics:

relay.get()

This API function allows the client to invoke any command supported by the specified node directly on our proxy instance.

Inputs

No input value.

Outputs

Get the current value of the specified node.

Example

```
>>> relay.get()
send: 'POST /XMLRPCv2/RNA/interfaces/relay1 HTTP/1.0\r\nHost: beggar\r\nAuthorization:
Basic bXB4YWRtaW46bXB4YWRtaW4=\r\nUser-Agent: xmlrpclib.py/1.0.1 (by
www.example.com)\r\nContent-Type: text/xml\r\nContent-Length: 97\r\n\r\n'
send: "<?xml
version='1.0'?>\n<methodCall>\n<methodName>get</methodName>\n<params>\n</params>\n</method
Call>\n"
reply: 'HTTP/1.0 200 OK\r\n'
header: Date: Wed, 08 Jul 2009 01:36:06 GMT
header: Content-Length: 121
header: Content-Type: text/xml
header: Connection: close
header: Server: WebServer/unofficial(release): 3.0.2 build 3
body: "<?xml
version='1.0'?>\n<methodResponse>\n<params>\n<param>\n<value><int>0</int></value>\n</param
>\n</params>\n</methodResponse>\n"
0
```

relay.set()

This API function allows the client to set the current value of the specified node.

Inputs

No input value.

Outputs

No return value

Example

```
>>> relay.set(1)
```

```
Output of the request content follows.send: 'POST /XMLRPCv2/RNA/interfaces/relay1
HTTP/1.0\r\nHost: beggar\r\nAuthorization: Basic bXB4YWRtaW46bXB4YWRtaW4=\r\nUser-Agent:
xmlrpclib.py/1.0.1 (by www.example.com)\r\nContent-Type: text/xml\r\nContent-Length:
142\r\n\r\n'
send: "<?xml
version='1.0'?>\n<methodCall>\n<methodName>set</methodName>\n<params>\n<param>\n<value><in
t>1</int></value>\n</param>\n</params>\n</methodCall>\n"
reply: 'HTTP/1.0 200 OK\r\n'
header: Date: Wed, 08 Jul 2009 01:36:29 GMT
header: Content-Length: 115
header: Content-Type: text/xml
header: Connection: close
header: Server: WebServer/unofficial(release): 3.0.2 build 3
body: "<?xml
version='1.0'?>\n<methodResponse>\n<params>\n<param>\n<value><nil/></value>\n</param>\n</p
arams>\n</methodResponse>\n"
# set() commands have no explicit return value.
>>> # Confirm that the value has been changed.
>>> relay.get()
# Output of the request content follows.
send: 'POST /XMLRPCv2/RNA/interfaces/relay1 HTTP/1.0\r\nHost: beggar\r\nAuthorization:
Basic bXB4YWRtaW46bXB4YWRtaW4=\r\nUser-Agent: xmlrpclib.py/1.0.1 (by
www.example.com)\r\nContent-Type: text/xml\r\nContent-Length: 97\r\n\r\n'
send: "<?xml
version='1.0'?>\n<methodCall>\n<methodName>get</methodName>\n<params>\n</params>\n</method
Call>\n"
reply: 'HTTP/1.0 200 OK\r\n'
header: Date: Wed, 08 Jul 2009 01:36:46 GMT
header: Content-Length: 121
header: Content-Type: text/xml
header: Connection: close
header: Server: WebServer/unofficial(release): 3.0.2 build 3
body: "<?xml
version='1.0'?>\n<methodResponse>\n<params>\n<param>\n<value><int>1</int></value>\n</param
>\n</params>\n</methodResponse>\n"
# The actual value returned is below.
1
```

The following example shows the verbose output turned off to demonstrate the actual exchange clearly:

```
>>> timenode =
xmlrpclib.ServerProxy("https://mpxadmin:mpxadmin@beggar/XMLRPCv2/RNA/services/time")
>>> timenode.hasattr("get")
True
>>> timenode.get()
1247017336.005939
>>> timenode.hasattr("set")
False
>>> timenode.getattr("name")
'time'
>>> timenode.configuration()
{'debug': '0', 'enabled': '1', 'name': 'time', 'parent': '/services'}
>>>
```

The following example shows what happens when you try to invoke a non-existent function:

Note You should observe how the queries using hasattr() indicate which methods may or may not be invoked on the current node.

```
>>> timenode.hasattr("set")
```

```
False
>>> timenode.set(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File
"/System/Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/xmlrpclib.py",
line 1147, in __call__
    return self.__send(self.__name, args)
  File
"/System/Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/xmlrpclib.py",
line 1437, in __request
    verbose=self.__verbose
  File
"/System/Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/xmlrpclib.py",
line 1201, in request
    return self._parse_response(h.getfile(), sock)
  File
"/System/Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/xmlrpclib.py",
line 1340, in _parse_response
    return u.close()
  File
"/System/Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/xmlrpclib.py",
line 787, in close
    raise Fault(**self._stack[0])
xmlrpclib.Fault: <Fault 1:
'<exception>\n<module>exceptions</module>\n<class>exceptions.AttributeError</class>\n<str>
"\'Time\' object has no attribute \'set\'"</str>\n</exception>'>
```

The following example shows how the user can catch the error by using a try/except block:

```
>>> try:
...   timenode.set(1)
... except Exception, error:
...   print "Error occured: %r" % error
...
Error occured: <Fault 1:
'<exception>\n<module>exceptions</module>\n<class>exceptions.AttributeError</class>\n<str>
"\'Time\' object has no attribute \'set\'"</str>\n</exception>'>
```