XML-RPC

The poor man's SOAP

Copyright

XML-RPC

- Codified in April 1998
- An RPC method for providing web services
- As **simple** and **standard** as possible
  - Uses HTTP as the transport method
  - An XML-RPC call is an HTTP-POST request
  - Data is encoded in XML
    - The content of the call must be content-type 'text/xml'
    - The HTTP response will be content-type text/xml
      - The response will contain a single <methodresponse>

XML-RPC vs. SOAP

XML-RPC

Pros

- Simple
- Light (few resources consumed)

Cons

- Myriad Implementations
- Limited error handling
- Limited Data-types
- No introspection
- Positional Parameters

SOAP

Pros

- Fast
- Consistent
- Introspection
- Named Parameters
- Complex Data-types
- Rich Toolchain

Cons

- Dependencies
- Complex

XML-RPC Data Types

Integer	<i4> or <int>
Boolean	<boolean>
ASCII String	<string>
Float	<double>
Date & Time	<dateTime.iso8601>
Encoded Binary	<base64>

- Leading zeros on integer responses are dropped
- The sign on numeric valus must precede the value (-100 vs. 100-)
- No representation for positive or negative infinity
- No representation for “not a number”
- No representation for NULL
- Whitespace is not allowed in numeric values
- String character “<” is escaped as “&lt”, “>” as “&gt”
- String characyer “&” is escaped as “&”

An XML-RPC Structure

Structures contain an arbitrary number of members

Each member must contain a name and a value.

```
<struct>
  <member>
        <name>OID</name>
        <value><string>1.3.6.1.4.1.5322.10.1.1</string></value>
   </member>
  <member>
        <name>Name</name>
        <value><string>krb5PrincipalName</string></value>
   </member>
</struct>
```

- Order of keys may not be maintained.
- Keys must be strings.
- Structures can be nested.

An XML-RPC Array

```
<array>
  <data>
         <value><string>1.3.6.1.4.1.5322.10.1.1</string></value>
         <value><string>krb5PrincipalName</string></value>
         <value><int>-31</int></value>
    </data>
</array>
```

- An array may not maintain the order of the values.
- An array may contain mixed data types.
- Arrays can be nested.

An XML-RPC Call

Function name

```
<?xml version="1.0"?>
     <methodCall>
          <methodName>get.Temperature</methodName>
               <params>
                    <param>
                         <value><string>Detroit</string></value>
                         <value><string>Michigan</string></value>
                    </param>
               </params>
          </methodCall>
```

Parameters

XML-RPC Return Values

- If the XML-RPC succeeds
  - <methodresponse> will contain a single <params>
    - The <params> will contain a single <param>
      - The <param> will contain a single <value>

```
<methodresponse>
    <params>
        <param>
            <value>
                <int>100031</int>
            </value>
        </param>
    </params>
</methodresponse>
```

XML-RPC Error Example

Error result is a XML structure.

```
<methodresponse>
     <fault>
          <value>
               <struct>
                    <member>
                         <name>faultCode</name>
                         <value><int>100</int></value>
                    </member>
                     <member>
                         <name>faultString</name>
                         <value><string>No Route to Host</string></value>
                    </member>
               </struct>
          </value>
     </fault>
</methodresponse>
```

XML-RPC Error Response

- If the XML-RPC call fails...
  - <methodresponse> will contain a single <fault>
    - The <fault> will contain a single <value>
      - The <value> will contain a single two member <struct>
        - The first member of the struct is the faultCode
          - Type integer
        - The second member of the struct is the faultstring
          - Type string
    - These is no standard for fault codes, although some services use an equivalent HTTP, to the extent one exists, error code as a faultCode.

xmlrpclib

simple client

```
#!/usr/bin/env python
import xmlrpclib
server = xmlrpclib.Server('http://adam:******@localhost/zidestore/so/adam/')
criteria1 = { }
criteria1['conjunction'] = 'OR'
criteria1['key'] = 'email2'
criteria1['value'] = '%handling%'
criteria1['expression'] = 'ILIKE'
query = [ criteria1,  ]
flags = { 'limit' : 150,
          'revolve': 'NO' }
try:
  result = server.zogi.searchForObjects('Enterprise', query, 0, flags)
  for enterprise in result:
    ...
except xmlrpclib.Fault, err:
  print "Fault code: %d" % err.faultCode
  print "Fault string: %s" % err.faultString
except xmlrpclib.ProtocolError, err:
  print "Error code: %d" % err.errcode
  print "Error message: %s" % err.errmsg
```

xmlrpclib

client with transport

```
#!/usr/bin/env python
import xmlrpclib, pprint
import orgWhitemiceXmlRpc
transport = orgWhitemiceXmlRpc.Transport()
transport.credentials = ("adam", "*********")
#transport.set_proxy("squid.mormail.com:3128")
server = xmlrpclib.ServerProxy("http://opengroupware/zidestore/so/adam/",
                               transport=transport)
criteria1 = { }
criteria1['conjunction'] = 'OR'
criteria1['key'] = 'email2'
criteria1['value'] = '%handling%'
criteria1['expression'] = 'ILIKE'
query = [ criteria1,  ]
flags = { 'limit' : 150,
          'revolve': 'NO' }
try:
  result = server.zogi.searchForObjects('Enterprise', query, 0, flags)
  for enterprise in result:
    ...
except xmlrpclib.Fault, err:
  print "Fault code: %d" % err.faultCode
...
```

xmlrpclib

server

```
#!/usr/bin/python
from SimpleXMLRPCServer import SimpleXMLRPCServer
from SimpleXMLRPCServer import SimpleXMLRPCRequestHandler

class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)
server = SimpleXMLRPCServer(("localhost", 8000),
                            requestHandler=RequestHandler)
class RPCFunctions:
    def div(self, x, y):
        return x // y
server.register_instance(RPCFunctions())
server.serve_forever()
```

```
#!/usr/bin/python
import xmlrpclib

s = xmlrpclib.ServerProxy('http://localhost:8000')
print s.div(5,2)  # Returns 5//2 = 2
```

xmlrpclib

DateTime & Binary Data

Dates:

```
#Get a datetime object
today = datetime.datetime.today()
# Make an XML-RPC datetime
xmlrpctoday = xmlrpclib.DateTime(today)
# Make a datetime from an XML-RPC datetime
print datetime.datetime.strptime(xmlrpctoday.value,
                              "%Y%m%dT%H:%M:%S")
```

Binary Server:

```
def python_logo():
     with open("python_logo.jpg") as handle:
         return xmlrpclib.Binary(handle.read())
```

Binary Client:

```
with open("fetched_python_logo.jpg", "w") as handle:
    handle.write(proxy.python_logo().data)
```

XML-RPC Services

- zOGI : OpenGroupware XML-RPC ZideStore provider
  - http://code.google.com/p/zogi/
- Address Miester
  - http://www.addressmeister.com/webservice_integration.htm
- xmlBlaster
  - http://www.xmlblaster.org/
- xml-rpc.net – XML-RPC assembly for Mono/.NET
  - http://www.xml-rpc.net/