
CS 696

Introduction to Grid Computing:
Lecture #7

Mary Thomas

San Diego State

Thursday, 8-Feb-07 (updated 10-Feb-07)

Announcements

- Today:
 - PyXML
 - Great tutorial:
 - http://www.boddie.org.uk/python/XML_intro.html
 - Client/Server
 - Using Telnet:
 - <http://www.faqs.org/rfcs/rfc854.html>
 - <http://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html#sec4.3>
 - SOAP
- HW#2 has been posted
 - 3 problems (due on different dates)

HW Prob 2.1: Python Client-Server

- Download PythonWare archive from course website:
 - http://www-rohan.sdsu.edu/faculty/mthomas/courses/spring07/cs696/downloads/progxmlrpc_examples.zip
- Note: Python 2.2 contains xmlrpclib
- You will work with chapt6 example
 - Covered partially in class today
 - Copy *.txt file to *.py
 - Fix bugs and get client & server examples working

HW Prob 2.2: XML Parsing

- Write a python script to create the Shared Math Web Service XML document
 - Start with example in lectures
 - Must include elements for methods for the following:
 - Ops: add, sub, div, mult, sqrt, exp
 - DataTypes: integer, floating point
 - Args of function should reflect operator
 - Eg myAdd(num1, num2), returns num3

HW Prob 2.3: XML Parsing

- Write a python script to create the Shared Math Web Service XML document
 - Start with example in lectures
 - Must include elements for methods for the following:
 - Ops: add, sub, div, mult, sqrt, exp
 - DataTypes: integer, floating point
 - Args of function should reflect operator
 - Eg myAdd(num1, num2), returns num3

Python and XML: PyXML

- Collection of libraries to process XML with Python:
 - xmlproc: a validating XML parser.
 - Expat: a fast non-validating parser.
 - sgmlop: C helper module; speed-up xmllib.py and sgmlib.py by a factor of 5.
 - PySAX: SAX 1/SAX2 libs + drivers for most parsers.
 - 4DOM: A fully compliant DOM Level 2 implementation
 - javadom: An adapter from Java DOM implementations to the standard Python DOM binding.
 - pulldom: supports lazy instantiation of nodes.
 - marshal: a module with several options for serializing Python objects to XML, including WDDX and XML-RPC.

PyXML installation

- Download from:
 - From <http://pyxml.sourceforge.net/>
- Put in desired directory (e.g.
 - ~/mthomas/working/python/PyXML
- Then install:
 - Sudo python setup.py install
- This took me less than 10 minutes

Python Web Services: Simple ADD

```
>>> import xml.dom.ext
```

```
>>> import xml.dom.minidom
```

- >>> dir(xml.dom.minidom)
- ['Attr', 'AttributeList', 'CDATASection', 'CharacterData', 'Childless', 'Comment', 'DOMImplementation', 'DOMImplementationLS', 'Document', 'DocumentFragment', 'DocumentLS', 'DocumentType', 'EMPTY_NAMESPACE', 'EMPTY_PREFIX', 'Element', 'ElementInfo', 'EmptyNodeList', 'Entity', 'GetattrMagic', 'Identified', 'NamedNodeMap', 'NewStyle', 'Node', 'NodeList', 'Notation', 'ProcessingInstruction', 'ReadOnlySequentialNamedNodeMap', 'StringTypes', 'Text', 'TypeInfo', 'XMLNS_NAMESPACE', '__TupleType', '__builtins__', '__doc__', '__file__', '__name__', '_append_child', '_clear_id_cache', '_clone_node', '_do_pulldom_parse', '_get_StringIO', '_get_containing_element', '_get_containing_entref', '_get_elements_by_tagName_helper', '_get_elements_by_tagName_ns_helper', '_in_document', '_no_type', '_nodeTypes_with_children', '_nssplit', '_set_attribute_node', '_write_data', 'defproperty', 'domreg', 'getDOMImplementation', 'parse', 'parseString', 'xml']

```
>>> import xml.dom.ext
```

- ```
>>> import xml.dom.ext
```
- ```
>>> dir(xml.dom.ext)
```
- ```
['Canonicalize', 'DOMException', 'FtDomException',
'GetAllNs', 'GetElementById',
'HTML_4_TRANSITIONAL_INLINE', 'IsDOMString', 'Node',
'NodeFilter', 'NodeTypeDict', 'NodeTypeToClassName',
'PrettyPrint', 'Print', 'Printer', 'ReleaseNode', 'SeekNss',
'SplitQName', 'StripHtml', 'StripXml', 'Visitor',
'XHtmlPrettyPrint', 'XHtmlPrint', 'XMLNS_NAMESPACE',
'XML_NAMESPACE', 'XmlSpaceState', '__builtins__',
'__doc__', '__file__', '__name__', '__path__', '_id_key',
'c14n', 're', 'string', 'sys', 'types']
```

# PyXML: Creating a Basic XML document

Note: these examples are examples of bad coupling: everything is hardcoded

```
import xml.dom.minidom
def get_a_document():
 doc = xml.dom.minidom.Document()
 business_element = doc.createElementNS("http://www.boddie.org.uk/paul/business", "business")
 doc.appendChild(business_element)
 return doc, business_element
def add_a_location(doc, business_element):
 location_element = doc.createElementNS("http://www.boddie.org.uk/paul/business", "location")
 business_element.appendChild(location_element)
 return location_element
```

```
>>> doc, business_element = get_a_document()
>>> doc.childNodes
[<DOM Element: business at 0x6d94e0>]
>>> doc.childNodes[0].localName
'business'
>>> location_element = add_a_location(doc, business_element)
>>> doc.childNodes[0]
<DOM Element: business at 0x6d94e0>
>>> doc.childNodes[0].childNodes
[<DOM Element: location at 0x6d9530>]
>>> doc.childNodes[0].childNodes[0].namespaceURI
'http://www.boddie.org.uk/paul/business'
>>> doc.childNodes[0].childNodes[0].localName
'location'
```

# PyXML: adding Elements

```
def add_surroundings(doc, location_element):
 surroundings_element = doc.createElementNS("http://www.boddie.org.uk/paul/business",
 "surroundings")
 location_element.appendChild(surroundings_element)
 description = doc.createTextNode("A quiet, scenic park with lots of wildlife.")
 surroundings_element.appendChild(description)
 return surroundings_element
def add_more_surroundings(doc, surroundings_element):
 description = doc.createTextNode(" It's usually sunny here, too.")
 surroundings_element.appendChild(description)
```

Adding two more functions

Normalize will  
combine nodes

```
>>> add_more_surroundings(doc, surroundings_element)
>>> surroundings_element.childNodes
[<DOM Text node "A quiet, s...">, <DOM Text node " It's usua...">]
>>> surroundings_element.childNodes.nodeValue
Traceback (most recent call last):
 File "<stdin>", line 1, in ?
AttributeError: 'NodeList' object has no attribute 'nodeValue'
>>> surroundings_element.childNodes
[<DOM Text node "A quiet, s...">, <DOM Text node " It's usua...">]
>>> surroundings_element.normalize()
>>> surroundings_element.childNodes
[<DOM Text node "A quiet, s...">]
>>> surroundings_element.childNodes[0].nodeValue
"A quiet, scenic park with lots of wildlife. It's usually sunny here, too."
```

# PyXML: adding Attributes to Elements

```
def add_building(doc, location_element):
 building_element = doc.createElementNS("http://www.boddie.org.uk/paul/business", "building")
 location_element.appendChild(building_element)
 return building_element
```

Adding two more  
functions

```
def name_building(building_element):
 building_element.setAttributeNS("http://www.boddie.org.uk/paul/business",
 "business:name", "Ivory Tower")
```

```
>>> name_building(building_element)
>>> building_element.getAttributeNS("http://www.boddie.org.uk
/paul/business", "name")
'Ivory Tower'
```

# PyXML: the XML

```
import xml.dom.ext
import xml.dom.minidom

def write_to_screen(doc):
 xml.dom.ext.PrettyPrint(doc)
```

```
>>> write_to_screen(doc)
<?xml version='1.0' encoding='UTF-8'?>
<business xmlns='http://www.boddie.org.uk/paul/business'
xmlns:business='http://www.boddie.org.uk/paul/business'>
 <location>
 <surroundings>A quiet, scenic park with lots of wildlife. It's usually sunny
here, too.</surroundings>
 <building business:name='Ivory Tower'/>
 </location>
</business>
>>>
```

# PyXML: Importing XML from file

```
import xml.dom.minidom
def import_document(name="ace1Math.xml"):
 return xml.dom.minidom.parse(name)
```

This is the XML document used by the DummyServer and DummyClient Java examples from previous lecture

```
>>> doc2 = get_a_document()
>>> write_to_screen(doc2)
<?xml version='1.0' encoding='UTF-8'?>
<methodCall>
<methodName>pyServer.addDb1</methodName>
<params>
 <param>
 <value>
 <double>5.0</double>
 </value>
 </param>
 <param>
 <value>
 <double>3.0</double>
 </value>
 </param>
</params>
</methodCall>
```

# Python XML-RPC Server: `acelMathServer.py`

```
import SocketServer, BaseHTTPServer
import xmlrpclib, sys
import xml.dom.minidom
import xml.dom.ext
from types import *

class RequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):
 def do_POST(self):
 try:
 # get arguments
 data = self.rfile.read(int(self.headers["content-length"]))
 params, method = xmlrpclib.loads(data)
 try: # generate response
 response = self.call(method, params)
 if type(response) != type(()):
 response = (response,)
 except xmlrpclib.Fault, faultobj:
 # generated response was a Fault
 response = xmlrpclib.dumps(faultobj)
 except: # report exception back to server
 response = xmlrpclib.dumps(xmlrpclib.Fault(1, "%s:%s" % (sys.exc_type, sys.exc_value)))
 else:
 # no exception occurred:# send method's return value back to server
 response = xmlrpclib.dumps(response, methodresponse=1)
```

# Python XML-RPC Server: `acelMathServer.py` (cont.)

```
except:
 # internal error, report as HTTP server error
 self.send_response(500)
 self.end_headers()
else:
 # no internal error: send back a "success" response
 self.send_response(200)
 self.send_header("Content-type", "text/xml")
 self.send_header("Content-length", str(len(response)))
 self.end_headers()
 self.wfile.write(response)

 # shut down the connection (from Skip Montanaro)
 self.wfile.flush()
 self.connection.shutdown(1)
```

# Python XML-RPC Server: `acelMathServer.py` (cont.)

```
def call(self, method, params):
override this method to implement RPC methods
print "CALL", method, params
return params

def call(self, meth_name, arg_tuple):
 myAPI = ("addDbI", "addInt")
 if meth_name in myAPI:
 func_obj = eval("do_" + meth_name)
 return apply(func_obj, arg_tuple)
 else:
 raise xmlrpclib.Fault(123, "Unknown method name")

functions that implement XML-RPC API
def do_addDbI(dbl1=0, dbl2=0):
 dbl3=dbl1+dbl2
 return dbl3
```

# acelMathServer.py: MAIN

```
if __name__ == '__main__':
 # set up and run XML-RPC server process (daemon)
 host = "http://localhost:8000"
 p = host.split(':')
 port = int(p[2])
 print "starting server on port %s ..." % port
 # create server
 server = SocketServer.TCPServer(("", port), RequestHandler)
 # send startup message to console
 print "Mary's XML-RPC server listening on port %s ..." % port
 # start server
 server.serve_forever()
```

```
mthomas% python exmathserv.py &
[1] 28751
[gidget:cs696/xmlrpc/math] mthomas% starting server on port 8000 ...
Mary's XML-RPC server listening on port 8000 ...
[gidget:cs696/xmlrpc/math] mthomas% python exmathclient.py
Server Client: connecting to host: http://localhost:8000
localhost - - [08/Feb/2007 17:19:36] "POST /RPC2 HTTP/1.0" 200 -
Client calling acelMathServer function: addDbI(f1,f2)
adding 3.141600 + 450.000000 = 37.000000
```

## Python XML-RPC Client: `acelMathClient.py`

```
import xmlrpclib, sys
from types import *
import SocketServer, BaseHTTPServer
host = "http://localhost:8000"
print "Server Client: connecting to host: %s\n" % host
create object representing connection to server host
from xmlrpclib import Server
conn=Server(host)

class MyClass: # user-defined object
 def __init__(self): #this must be present
 pass
try:
 f1,f2 = 3.1416, 4.5e2 # floating-point number
 response = conn.addDbl(12.4, 24.67)
 f3 = int(response)
 print ("Call: acelMathServer: addDbl(f1,f2)")
 print "\tadding %f + %f = %f" % (f1,f2,f3)
except xmlrpclib.Fault, faultobj:
 print "Server error:", faultobj.faultCode
 print ">>> %s <<<" % faultobj.faultString
except:
 print "Client error: '%s/%s'" % (sys.exc_type,
 sys.exc_value)
```

```
[gidget:% python mathclient.py
Server Client: connecting: http://localhost:8000
Client calling acelMathServer function: addDbl(f1,f2)
adding 3.141600 + 450.000000 = 453.141600
```

---

# acelMathServer Client: interactive python session

```
>>> import SocketServer, BaseHTTPServer
>>> from xmlrpclib import Server
>>> conn=Server(host)
>>> f1=3.1316
>>> f2=4.5e2
>>> response = conn.addDbl(12.4, 24.67)
localhost - - [08/Feb/2007 16:59:23] "POST /RPC2 HTTP/1.0" 200 -
>>> f3=int(response)
>>> print "\tadding %f + %f = %f" % (f1,f2,f3)
 adding 3.141600 + 450.000000 = 453.141600
```

# XML-RPC Request Format

payload



HEADER... .

```
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
 <methodName>dummy.addFloat</methodName>
 <params>
 <param>
 <value><float>3.14156</float></value>
 </param>
 <param>
 <value><float>450.0</float></value>
 </param>
 </params>
</methodCall>
```

Hint: to see the actual XML generated from web services libraries, you need to identify which library method will allow you see the 'raw' input and output streams being parsed by the server.

# Telnet to XML-RPC Service: Error

```
[gidget:~] mthomas% telnet localhost 8000
Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
POST /RPC HTTP/1.0
Host: localhost:8000
Content-Type: text/xml
Content-length: 231
[CRLF]
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
<methodName>addFloat</methodName>
<params>
<param>
<value><float>3.14156</float></value>
</param>
<param>
<value><float>450.0</float></value>
</param>
</params>
</methodCall>
```

```
<?xml version='1.0'?>
<methodResponse>
<fault>
<value><struct>
<member>
<name>faultCode</name>
<value><int>1</int></value>
</member>
<member>
<name>faultString</name>
<value><string>exceptions.UnboundLocalError:local
variable 'method'
referenced before assignment</string></value>
</member>
</struct></value>
</fault>
</methodResponse>
Connection closed by foreign host.
```

# Telnet to XML-RPC Service: Successful request/response

## CLIENT REQUEST

```
POST /RPC HTTP/1.1
Host: localhost:8000
Content-Type: text/xml
Content-length: 235
[CRLF]
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
<methodName>addFloat</methodName>
<params>
<param>
<value><float>3.14156<float></value>
</param>
<param>
<value><float>450.0</float></value>
</param>
</params>
</methodCall>
```

Errors were due to these lines.  
Also, note that request sent float, while server parsed it into a double.

## SERVER RESPONSE

```
HTTP/1.0 200 OK
Server: BaseHTTP/0.3 Python/2.3.5
Date: Sat, 10 Feb 2007 20:51:41 GMT
Content-type: text/xml
Content-length: 144

<?xml version='1.0'?>
<methodResponse>
<params>
<param>
<value><double>453.14159999999998</double></value>
</param>
</params>
</methodResponse>
```

---

# Simple Object Access Protocol (SOAP)

- A SOAP message is fundamentally a one-way transmission between SOAP nodes, from a SOAP sender to a SOAP receiver
- SOAP messages are expected to be combined by applications to implement more complex interaction patterns:
  - request/response
  - multiple, back-and-forth "conversational" exchanges
  - etc.

---

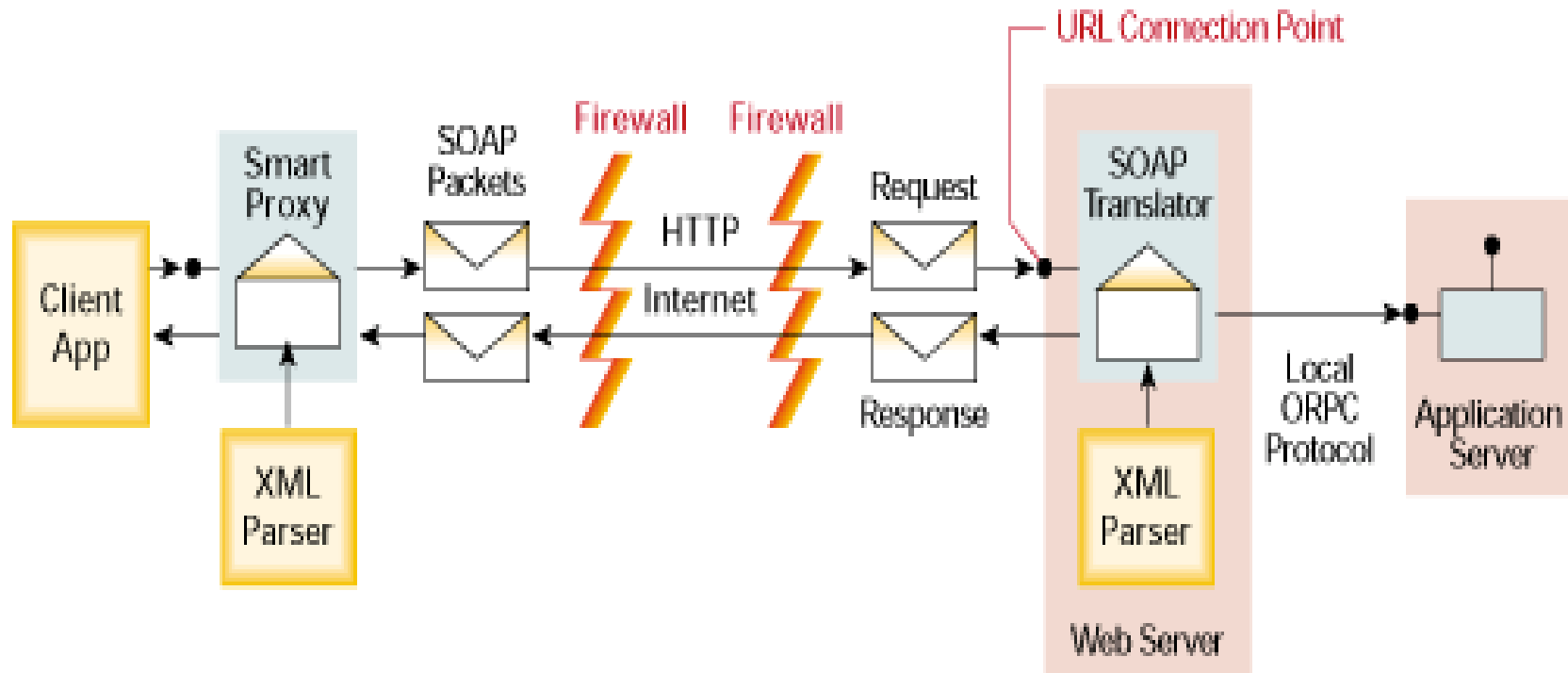
# SOAP - Acronym for

- **Simple**: Transporting XML structured messages across internet using HTTP
- **Object**: transportation of COM objects
  - Common Object Model (COM): open software architecture from DEC, Microsoft, allowing interoperation between ObjectBroker and OLE
  - Microsoft evolved COM into DCOM.
- **Access** - a philosophy: services will be easier to deploy when binding them to common protocols (HTTP)
  - Most firewalls already pass through web page data
- **Protocol**: SOAP is an XML based protocol used to exchange distributed data over HTTP
  - ~~Origins in RPC~~

# SOAP vs DCOM/CORBA

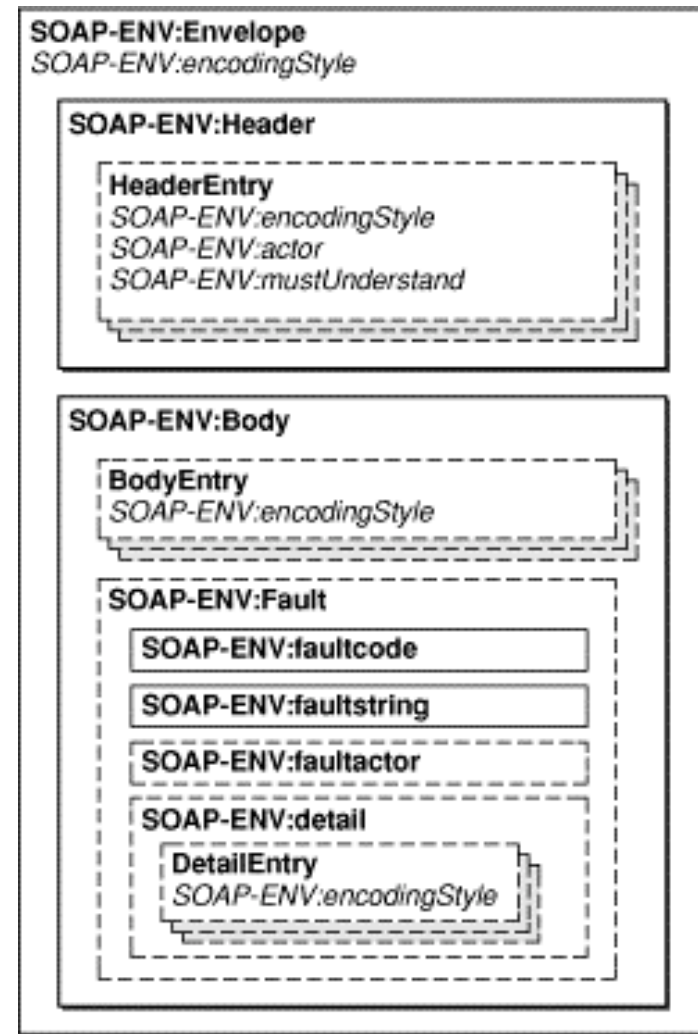
- SOAP is a protocol:
  - mandates how a method call transfers over the wire.
- SOAP model follows standard Web and distributed apps:
  - Stateless transactions provide client with data needed for task:
    - Get data from Web server via method call, use data, send updated data back to server in another stateless remote call.
- SOAP uses single function or method invocations
  - Does not maintain state to an object between calls.
  - client makes single method calls, one/HTTP request.
- DCOM/CORBA:
  - client creates a persistent connection to the object to perform multiple property accesses and method calls.
- Both support stateful remote connections –
  - SOAP typically does not
    - could be implemented with server side state management of an application

# SOAP Architecture



# SOAP Components and Elements

- Components:
  - Formatting conventions
  - Transport/protocol binding
  - Encoding rules
  - RPC mechanism
- Elements:
  - **Envelope**: Required
  - **Header** [Optional] - use:
    - Transaction data
  - **Body**: Required, use:
    - Method call and its parameters



---

# SOAP Syntax Rules

- A SOAP message **MUST** be encoded using XML
- A SOAP message **MUST** use the SOAP Envelope namespace
- A SOAP message **MUST** use the SOAP Encoding namespace
- A SOAP message must **NOT** contain a DTD reference
- A SOAP message must **NOT** contain XML Processing Instruction

# Skeleton SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
 ...
 ...
</soap:Header>
<soap:Body>
 ...
 ...
 <soap:Fault>
 ...
 ...
 </soap:Fault>
</soap:Body>
</soap:Envelope>
```

# SOAP Envelope

- Root element of a SOAP message.
- Defines the XML document as a SOAP message
- The xmlns:soap Namespace:
  - message must have Envelope element associated with W3C namespace:
    - <http://www.w3.org/2001/12/soap-envelope>

```
<soap:Envelope

 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"

 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
```

# SOAP Header

- Elements in SOAP Header define how a recipient should process the SOAP message
- Header elements must have attributes:
  - 3 attributes in namespace ("http://www.w3.org/2001/12/soap-envelope")
- *actor*: used to address Header element to endpoint
- *mustUnderstand true*: receiver processing Header must recognize element
- *encodingStyle*

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
<m:Trans
xmlns:m="http://www.w3schools.com/transaction/"
soap:actor="http://www.w3schools.com/appml/">
234
</m:Trans>
</soap:Header>
```

# SOAP Body

## REQUEST:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap
-envelope"
soap:encodingStyle="http://www.w3.org/2001
/12/soap-encoding">
<soap:Body>
 <m:GetPrice
 xmlns:m="http://www.w3schools.com/pric
 es">
 <m:Item>Apples</m:Item>
 </m:GetPrice>
</soap:Body>
</soap:Envelope>
```

- Body is XML data
- User namespace defined
- Fault: defined by SOAP standard for errors

# SOAP HTTP Binding

- HTTP Binding
- HTTP + XML = SOAP:
  - Content-type
- Protocol:
  - Client sends request
  - Server responds OK

```
POST /item HTTP/1.1
Host: 189.123.345.239
Content-Type: MIMEType;
 charset=character-
 encoding
Content-Length: 200

200 OK
Content-Type: text/plain
Content-Length: 200
```

# SOAP Request/Response Example

```
POST /InStock HTTP/1.1
Host: www.stock.org
Content-Type: application/soap+xml;
 charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/1
 2/soap-envelope"
soap:encodingStyle="http://www.w3.or
 g/2001/12/soap-encoding">

 <soap:Body
 xmlns:m="http://www.stock.org/st
 ock">
 <m:GetStockPrice>
 <m:StockName>IBM</m:StockName>
 </m:GetStockPrice>
 </soap:Body>

</soap:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: application/soap;
 charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/1
 2/soap-envelope"
soap:encodingStyle="http://www.w3.or
 g/2001/12/soap-encoding">

 <soap:Body
 xmlns:m="http://www.stock.org/st
 ock">
 <m:GetStockPriceResponse>
 <m:Price>34.5</m:Price>
 </m:GetStockPriceResponse>
 </soap:Body>

</soap:Envelope>
```

---

## Useful Links

- Good Books:
  - D. Chappell, T. Jewell, “Java Web Services,” O’Reilly
  - Graham, et. Al., “Building Web Services with Java,” SAMS
  - E. Cerami, “Web Services Essentials,” O’Reilly
- W3C Schools Web Site:
  - <http://www.w3schools.com>
- Apache SOAP website