Notes on how to use WSDL and document/literal encoding with SOAP::Lite

SOAP:lite does not look at the WSDL, but the pyGridware, Axis, and wsCompile do. So one needs to create a wsdl that corresponds to what the SOAP:lite server does.

----> Document Literal encoding

SOAP Lite defaults to soap rpc encoding. WS1-BP dictates document/literal. In document literal only the elements are sent over the wire. The message names and part names not used. You can specify document/literal encoding in a SOAP:lite client but it does not seem to get put in the response message, however it does not seem to matter to a pyGridware server.
e.g.

```
my $method = SOAP::Data -> name ('createReservation')
#    -> encodingStyle ('http://xml.apache.org/xml-soap/literalxml')
     -> attr ({xmlns => 'http://oscars.es.net/OSCARS/Dispatcher'});
The encoding stye in the message was still
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/
```

----> soap actions

SOAP::Lite server expects soap actions of the form uri/classname/method/

```
SOAP:Lite Client
  Specifying the soap action (taken from the WSDL)
# do not forget to specify the soapaction (on_action)
# you will find it in the wsdl,
# uri is the target namespace in the wsdl - doesn't seem to be used by client
# proxy is the endpoint address
my $soap = SOAP::Lite
   -> uri('http://oscars.es.net/OSCARS/')
   -> on_action( sub { return '"http://oscars.es.net/OSCARS/createReservation"'
} )
   -> proxy('http://localhost:8090/OSCARS');
```

The following worked

```
my $soap = SOAP::Lite
   -> uri('http://localhost:8090/foo')     # package  name
   -> proxy ('http://localhost:8090/OSCARS')    # script name
   -> on_action( sub { return '"http://oscars.es.net/OSCARS/ping"'});
```

The only thing that mattered was that the proxy pointed to the right host, port and file (OSCARS.pm), and that the action part matched the namespace, the class name and the method name.

To work with a python or java server the soapaction in a perl client
must match what is in the wdsl.

------> Making a SOAP::Lite server send document/literal messages

```
The following perl code in the server sends the message xml
# <inputQuery>
#   <login>...</login>
#   <password>...</password>
#   ....
#   <CoordType>...</CoordType>
# </inputQuery>

my $query =
  SOAP::Data
  ->name(inputQuery =>
    \SOAP::Data->value(   # the \SOAP causes the tag <InputQuery> to be included
      SOAP::Data->name(login => 'infobel'),
      SOAP::Data->name(password => 'test'),
      (...)
      SOAP::Data->name(CoordType => 'aeCTWGS')));

# make the call
my $result = $soap->call($method => $query);
```

----> SOAP::Lite client referring to the doc/literal return args

```
my $statusmes = $result->valueof('//SearchResponse/SearchResult/Status');
my $numrecs = $result->valueof('//SearchResponse/SearchResult/Result/NumRecs');
```

SOAP::Lite server referring to args

Parameters at the top level of the message get presented as an array to the server. This is not good for items of 1 or more instances. Items at lower levels get presented as hashes. A solution to multiple top-level arguments is to use the Server::Parameter class which will cause a SOM object to be added on the list of arguments. With the SOM there are several methods for retrieving arguments:

```
$som ->valueof(//createRequest/srcHost)  or

my($srcHost, $srcPort) = SOAP::Server::Parameters::byName([qw(srcHost srcPort)],
$envelope);
```

----> SOAP::Lite error handling

```
  print join ', ',
    $result->faultcode,
    $result->faultstring,
    $result->faultdetail;
```

----> Limitations of SOAP::Lite

SOAP::Lite is unable to pass a simple argument as the child =of a Top level element.
e.g.

```
 <soap:Body><ping1 type=xsd:string>input string</ping1></soap:Body>
```

It insists on either putting in a c_gensym tag or else having an explicit tag set. This requires a wsdl that specifies a sub element for any message

```
    <xsd:element name="testForwardResponse" type="tns:retMsg" />

   <xsd:complexType name="retMsg">
     <xsd:sequence>
        <xsd:element name="msg" type="xsd:string" />
     </xsd:sequence>
   </xsd:complexType>
```

Response names from a SOAP::lite server will be the input message name concatenated with Response. Thus if the operation is Ping the response message is PingResponse. So the WSDL must enforce this. This corresponds to rpc style of web services. It also picks up the name space of elements from the input message, so be sure that any uri's that are set match what is in the wsdl.

----> String enum messages

Strings that are an enum type need a custom deserializer on the perl server to deal with the type that pygridware and Axis sends. The simpler sub ab_enumType did not work because the typename was qualified with the namespace: {http://oscars.es.net/OSCARS}enumType which cannot be used as a subroutine name.

```
in the SOAP::Lite server
my $daemon = SOAP::Transport::HTTP::Daemon
  -> new (LocalPort => $port_number, Listen => 5, Reuse => 1)
  -> deserializer(MyDeserializer->new)
  -> dispatch_to('OSCARS');

package MyDeserializer;

use SOAP::Lite;
@MyDeserializer::ISA = 'SOAP::Deserializer';
```

```
sub typecast {

    my ($self, $val, $name, $attrs, $kids, $type ) = @_;
    return $val if  $type && $type =~ /enumType/;
    return undef;
};

sub as_emumType {
         my($self, $value, $name, $type, $attr) = @_;
         return $value;
}
```

-----> elementFormDefault="qualified"

This line in the xml::schema tag speficies that all the names will have a namespace associated with them. This was required to make wscompile clients work with the perl server. In some cases, pyGridware and Axis clients the namespace did not need to be explictly stated in the msg, but it could not be stated and not match what was in the wsdl.

---> empty messages

While axis, pyGridware and perl will allow messages with no values, this is not accpeted by WS1-BP and wscompile. The recommended way to pass no argument is

```
     <xsd:element name="listReservations"
          type="tns:emptyArg"/>

     <xsd:complexType name="emptyArg">
         <xsd:sequence>
          <xsd:element name="msg" minOccurs = "0"
               type="xsd:string" >
          </xsd:element>
         </xsd:sequence>
     </xsd:complexType>
```

-→ xsd:choice

Is not supported by wscompile. Axis and pyGridware will produce a class that contains all of the choices. Use a sequence (of 0-1 instances) instead where the first item specifies the choice that has been used.

wscompile complains

```
warning: unsupported XML Schema feature: "xsd:choice" in component
{http://oscars.es.net/OSCARS}pingchoice, mapping it to
javax.xml.soap.SOAPElement
```

resTag type

```
<xsd:complexType name="resTag">
      <xsd:sequence>
            <xsd:element name="sTag" type="xsd:string" />
      </xsd:sequence>
</xsd:complexType>
```

the following code:

```
SOAP::Data->name(createResReply =>
 \SOAP::Data->value(
     SOAP::Data->name('tag')
             ->attr({'xmlns:tns' =>
                'http://oscars.es.net/OSCARS/Dispatcher'})
             ->type('tns:resTag')
             ->value(\SOAP::Data->name(sTag => "a1234")),
```

generates the message

<createResReply><tag xsi:type="tns:resTag" xmlns:tns="http://oscars.es.net/OSCARS/Dispatcher">
<sTag xsi:type="xsd:string">a1234</sTag></tag>
(…)
</createResReply>

But the java client complains that sTag is an unrecognized element

```
SOAP::Data->name(createResReply =>
 \SOAP::Data->value(
     SOAP::Data->name('tag')
             ->attr({'xmlns:tns' =>
                'http://oscars.es.net/OSCARS/Dispatcher'})
             ->type('tns:resTag')
             ->value(a1234"),
```

generates

<createResReply><tag xsi:type="tns:resTag" xmlns:tns="http://oscars.es.net/OSCARS/Dispatcher">
a1234</tag>

Which the java client accepts, but then createResReply.getTag().getSTag() is null
and createResReply.getTag().toString is net.es.oscars.OSCARS.Dispatcher.ResTag@1

When a java client sends a resTag type the Java code is

ResTag queryRequest = new ResTag();

queryRequest.setSTag("1234");

and the message is
<queryReservation xmlns="http://oscars.es.net/OSCARS/Dispatcher">
<sTag>1234</sTag></queryReservation>