

Web Services



UW Extension
Perl and the World Wide Web

What are Web Services?

- Any piece of software that makes itself available over the Internet (or an intranet)
- Uses a standardized XML messaging system
- A public interface documented in a common XML grammar (WSDL)
- A simple mechanism to discover the service (UDDI)

What's new about Web Services?

- Three letters : XML
- RPCs weren't very portable
- XML is portable



XML-RPC

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <methodCall>
    <methodName>weather.getWeather</methodName>
    <params>
      <param><value>10016</value></param>
    </params>
  </methodCall>
```

XML-RPC

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodResponse>
  <params>
    <param>
      <value><int>65</int></value>
    </param>
  </params>
</methodResponse>
```

XML-RPC

- use Frontier::RPC

```
# Create a connection

my $server = Frontier::Client->new
  (url => 'http://localhost:8080/RPC2');

# Make calls on the server

my $result = $server-> call
  ('sample.sumAndDifference', 5, 3);
```

XML – RPC Types

- RPC enforces types, unlike perl
- Common types include boolean, date_time, base64, int, string
- For int and string, Frontier::Client will figure out the type for you.

```
# To enforce a type, use the server method for it.
```

```
my $var = $server->boolean(1);
```

XML – RPC Server

- Use Frontier::Daemon
- Starts its own HTTP server!
- Create a hash to name each Procedure available



```
my $methods = { 'sample.sumAndDifference' =>
                \&sumAndDifference,};
```

XML – RPC Server

- Start the HTTP Server
- Pick your port; start a “daemon”



```
Frontier::Daemon->new  
  (LocalPort => 8080, methods => $methods)  
  or die "Couldn't start HTTP server: $!";
```

XML - RPC

- More documentation:

<http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html>

SOAP::Lite

- (used to be) Simple Object Access Protocol
- The XML is pretty complex
- Uses XML namespaces heavily



SOAP Request

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://
schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/">

  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

SOAP Response

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://
schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://
schemas.xmlsoap.org/soap/encoding"/>

  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse
xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Using SOAP

- No need to parse all that XML yourself
- Use SOAP::Lite!
- Named for its ease-of-use, not functionality!
- <http://www.soaplite.com>

SOAP Client

- Use SOAP::Lite
- uri argument : namespace for the service
- Each server can support more than one webservice
- The 'uri' argument separates the services by namespace
- Proxy argument : URL of the service

SOAP Client

- Create a SOAP client object
- Call webservice method on the object to get result object
- Call `->result()` on result object to get single result
- Call `->paramsout()` to get remaining output parameters

SOAP Client Example

```
my $soapObj = SOAP::Lite->uri  
( 'http://www.ultravioletstudios.com/Demo' )  
->proxy( 'http://www.ultravioletstudios.com/cgi-  
bin/demoSOAPserver.cgi' );  
  
my $result = $soapObj->f2c(32)->result();
```



SOAP Server

- SOAP Servers can be CGI, mod_perl, or standalone HTTP daemon
- We'll use CGI

```
use SOAP::Transport::HTTP;  
SOAP::Transport::HTTP::CGI->dispatch_to('Demo')  
  ->handle();  
  
# Methods go in package 'Demo'
```

Object Access

- Place constructor (new) in the your service package
- OO Methods can be accessed most easily through the autodispatch feature of SOAP::Lite

Using SOAP::Lite autodispatcher

- Autodispatcher is especially useful when using an ObjectOriented Interface.

```
use SOAP::Lite +autodispatch => uri => $uri  
=> proxy => $proxy;
```

SOAP Examples

- Babelizer service
 - found on xmethods.net
 - two services calls
 - find supported languages
 - Translate text



BARNES & NOBLE.com
www.bn.com

SOAP Examples



- Barnes & Noble Book prices
- Given ISBN, get book price on BN.com
- See source code examples on class website

Finding Web Services

- Find webservices on <http://www.xmethods.net>
- You often have to do some sleuthing and experimenting to discover
- Whether the service requires parameter naming and typing
- What parameters the service requires
- Use Data::Dumper to discover the form of the output

Using Xmethods.Net

- Many of the services here require a license key
- Some services have mistakes in their WSDL
- By no means is xmethods.net comprehensive – there are lots of unadvertised webservices out there!



WSDL

- Web Services Description Language
- An attempt to standardize how to describe your webservice
- A computer program can read the description and discover how to talk to the service.
- The future : finding the webservice automatically:
UDDI
- Universal Description, Discovery, and Integration

WSDL In Practice

- Reading the WSDL can help you figuring out the parameters and return values.
- Some automatic WSDL interpreters do exist:
- SOAP::Lite to the rescue!

WSDL with SOAP::Lite

- SOAP::Lite can't read every WSDL, especially those with complex parameters, but it does a pretty good job
- More recent versions of SOAP::Lite do a better job
- .NET services are still pretty hard to interface with SOAP::Lite and Perl

SOAP::Lite with a WSDL

```
use SOAP::Lite;

my $service =
    SOAP::Lite->service("url of wsdl");

# Call methods on the $service object as before
```

SOAP one-liners!

- Using autodispatcher:

```
perl "-MSOAP::Lite service =>  
      'http://www.xmethods.net/sd/  
      StockQuoteService.wsdl'"  
  
-le "print getQuote('MSFT')"
```

SOAP and WSDL

- The WSDL analyzer at xmethods.net can help when you are unfamiliar with reading WSDLs



[Home](#) · [Tools](#) · [Implement](#)

A banner for StrikeIron with a blue background. It features the 'STRIKEIRON' logo on the left, the text 'Data as a Service. Hundreds of Services to Integrate Now!' in the center, and a yellow 'FREE TRIAL' badge on the right.

A banner for xignite with a dark background and a red starburst. It features the 'xignite' logo, the text '#1 in Financial Web Services' and 'MASH'EM UP!', and a list of services: 'Quotes, Forex, Rates, Metals, Financials, International, News, SEC Filings, etc...'. A 'Free Trial' badge is also present.

A banner for CDYNE CORPORATION with a white and green background. It features the 'CDYNE CORPORATION' logo, the text 'Postal Address Verification Web Service', and a small image of a document on the right.

Fortune Web Service

WSDL

<http://www.doughughes.net/WebServices/fortune/fortune.cfc?wsdl> Analyze WSDL | View RPC Proc

SOAP and WSDL

- Find the *operation* you want to use

```
<wsdl:portType name="fortune">
  <wsdl:operation name="getFortune"
    parameterOrder="topics minLength maxLength">
    <wsdl:input name="getFortuneRequest"
      message="impl:getFortuneRequest"/>
    <wsdl:output name="getFortuneResponse"
      message="impl:getFortuneResponse"/>
    <wsdl:fault name="CFCInvocationException"
      message="impl:CFCInvocationException"/>
  </wsdl:operation>
```

SOAP and WSDL

- Find the *message* for the *input* part of the operation

```
<wsdl:message name="getFortuneRequest">  
  <wsdl:part name="topics" type="xsd:string"/>  
  <wsdl:part name="minLength" type="xsd:double"/>  
  <wsdl:part name="maxLength" type="xsd:double"/>  
</wsdl:message>
```

SOAP and WSDL

- The *operation* is the method you call on the service object. The parts of the *message* are the arguments.

```
$service->getFortune  
(SOAP::Data->value  
  (SOAP::Data->name("topics")->value  
    ("simpsonshomer"),  
    SOAP::Data->name("minLength")  
      ->type(double => 1.0),  
    SOAP::Data->name("maxLength")  
      ->type(double => 500.0)));
```

SOAP and WSDL

- Notice that all the arguments are wrapped in a SOAP::Data (to wrap the array)

```
$service->getFortune  
(SOAP::Data->value  
  (SOAP::Data->name("topics")->value  
    ("simpsonshomer"),  
    SOAP::Data->name("minLength")  
      ->type(double => 1.0),  
    SOAP::Data->name("maxLength")  
      ->type(double => 500.0)));
```

SOAP and WSDL

- Also notice every item uses `SOAP::Data->name->value` to give both a name and a value to the argument.

```
$service->getFortune
(SOAP::Data->value
  (SOAP::Data->name("topics")->value
    ("simpsonshomer"),
    SOAP::Data->name("minLength")
      ->type(double => 1.0),
    SOAP::Data->name("maxLength")
      ->type(double => 500.0)));
```

SOAP and WSDL

- Finally notice that non-string arguments have to be typed (use `->type` instead of `->value`)

```
$service->getFortune  
(SOAP::Data->value  
  (SOAP::Data->name("topics")->value  
    ("simpsonshomer"),  
    SOAP::Data->name("minLength")  
      ->type(double => 1.0),  
    SOAP::Data->name("maxLength")  
      ->type(double => 500.0)));
```

SOAP and WSDL

- Getting results - use XPath syntax
- Simplest : just // and name of the node you want.

```
my $response = $service->call();  
print $response->valueof("//getFortuneReturn"), "\n";
```

Using WSDL With SOAP

- Amazon webservicess example (using WSDL)
- You need a developer token
- apply online at <http://www.amazon.com/webservices>

Using WSDL With SOAP

- Download the “SDK”
- The SDK has good description of a very large variety of methods and services
- The SDK contains only minimal direction on accessing the service, so using the WSDL is much easier than SOAP RPC calls.

Amazon Webservices

- Amazon webservices require parameters wrapped in object named similar to the service call. Use SOAP::Data:

```
$service->ItemSearch  
  (SOAP::Data->value  
    (SOAP::Data->name('SubscriptionId')  
      ->value($subscription_id),  
    SOAP::Data->name('Request')  
      ->value( \SOAP::Data->value  
        ( SOAP::Data->name('Keywords')  
          ->value($keywords),  
          SOAP::Data->name('SearchIndex')  
            ->value($index))))));
```

Using webservices

- <http://www.dealazon.com/dashboard/>
- <http://www.oxus.net/amazon/>
- <http://www.ahding.com/cheapgas/>
-