When implementing multi-tier software applications, you need to be able to make calls from programs running on Machine A to programs running on Machine B. For example, you might have a module on the business-logic layer that calculates the profit or loss of a certain business unit during a certain time period. In order to make that calculation, the logic module needs to make requests of one or more programs on the accessor layer. Assuming the accessor layer exists on another physical host, a mechanism needs to be devised to facilitate the passing of requests and responses between the two layers.

Various solutions for this problem exist, but none are as simple or as versatile as simple object access protocol (SOAP). SOAP allows the creation of Web services, which are standardized computing resources that take input from across a network and respond in an advertised way. Web services may or may not be significant to the future of the software industry at large—lots of software companies would like them to become so—but they certainly are convenient to the creation of multi-tier software applications.

SOAP is a specification for extensible markup language (XML) messages that travel between endpoints. It is a way for software programs to invoke other software programs (or parts of them) over the network. Just as you can make calls to locally defined functions from within an ordinary PHP program and expect to receive values in return, you can use SOAP to call functions that are defined in other programs, and in fact on completely separate machines. Just as is possible with calls to local functions, you can use SOAP to send one or more parameters along with a function invocation.

In terms of networking, SOAP messages travel on top of hypertext transport protocol (HTTP) or simple mail transport protocol (SMTP). That is, you must establish an HTTP or SMTP link to a remote resource before attempting to send SOAP messages there. In this book, we’ll use HTTP exclusively as a means of carrying SOAP messages. SMTP support is a newer addition to the SOAP specification, and it’s generally easier to get HTTP traffic
through firewalls and other network obstructions than SMTP traffic. Therefore, HTTP is usually the better choice, unless some specific condition requires that an application use SMTP to carry its SOAP messages.

A function can be exposed through a SOAP server, meaning it can accept SOAP messages from elsewhere on the network and return results in response. A single SOAP server object can expose lots of functions. These functions, when exposed, become Web services—computing resources that may be accessed over the network via Web protocols.

This chapter explains SOAP and Web services technologies, and gives you some idea of why you'd want to use them in your software architecture work. It also shows you the state of the art in SOAP packages for PHP, and demonstrates their use by means of some simple message-sending example applications.


### 4.1 Understanding SOAP

At the end of the day, Web pages are defined by hypertext markup language (HTML) documents, which are in turn a particular kind of text document. When we surf the Web, we're retrieving particular text files—streams of characters, if you prefer—which cause our browsers to behave in a certain way. Specifically, the streams of characters tell our browsers to display text, links, headers, and to retrieve other files (such as images), and to display them in particular locations.

The transport of the HTML text streams is accomplished by means of HTTP, which was covered in greater length in Chapter 3. One interesting aspect of HTTP is that it is both widely required and generally harmless, which means that routers are firewalls that are configured so that it's easier to get an HTTP request/response pair to pass between two given computers than any other kind of message exchange. Even if the two computers are separated by great distance and are connected only by the public Internet, it's usually possible for them to trade HTTP messages.
Therein lies a solution to the problem of interprocess communication. If the character streams provided by the HTTP request and response could be made to carry function calls rather than HTML pages meant for display, then the problem would be solved.

XML is a general-purpose markup language. It’s far less specialized than HTML and can be used for many different purposes. One of its applications is in formatting the messages that processes send back and forth as calls to remote methods and functions are made.

Because the messages are in XML, it’s not strictly necessary that the programs generating them all be written in the same language. As long as they all generate SOAP messages, it’s perfectly acceptable for a PHP program to make a request of, and receive a response from, a Web service written in C#, Java, or Perl. In fact, there’s an example of a PHP program making a call to a Web service implemented in Java in Chapter 10.

4.1.1 A SOAP Request

Suppose we have a program on the business-logic layer of a multi-tier application. It’s called updateCurrencies.php, and one of its functions needs to access a remote Web service (on a public-resource site called XMethods.net) to get the exchange rates of various currencies. The module in the business-logic layer can send out the names of two countries, and expects in return a real number (a float) that indicates the ratio of the value of the first country’s currency to the value of the second country’s currency.

In this situation, the Web service on XMethods.net would have to have a SOAP server active, and updateCurrencies.php would have to instantiate an instance of a SOAP client object. The SOAP client object would then send request messages, possibly with supplementary parameters, to the SOAP server, which would respond. Overall, the operation is very much like a call to a function in which a function name is invoked (and parameters possibly sent), and the function does some processing and returns a result. The only difference is in the amount of overhead (both locally and on the remote system) and in the intervening data network.

It is useful to look at the HTTP messages that pass between the machine running updateCurrencies.php and the Web service on XMethods.net (SOAP messages, again, ride on top of HTTP messages). The request from the business-logic layer to the remote resource looks like this:

```
POST /soap HTTP/1.0
User-Agent: NuSOAP v0.6
Host: services.xmethods.net
Content-Type: text/xml
Content-Length: 635
SOAPAction: ""

<?xml version="1.0"?>
<SOAP-ENV:Envelope
```
Some whitespace has been inserted into this excerpt for clarity, but the structure of the request is evident. First of all, this is an HTTP POST request, precisely the same as the HTTP POST requests covered in Chapter 3. It complies with the HTTP 1.0 specification in every way. The only unusual piece of this request is that it carries a payload. That payload is in the form of an XML document, which contains a call to a Web service called getRate() and two parameters (country1 and country2) for that Web service to process.

The next section examines the response that the XMethods.net Web service sends back.

4.1.2 A SOAP Response

The response, from XMethods.net to the business-logic layer, looks like this:

HTTP/1.1 200 OK
Date: Tue, 25 Nov 2003 07:38:56 GMT
Server: Electric/1.0
Content-Type: text/xml
Content-Length: 492
X-Cache: MISS from www.xmethods.net
Connection: close

<?xml version='1.0' encoding='UTF-8'?>

<soap:Envelope
xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
4.1 Understanding SOAP

```
xmllns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmllns:xsd='http://www.w3.org/2001/XMLSchema'
xmllns:soapenc='http://schemas.xmlsoap.org/soap/encoding/'
soap:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>

<soap:Body>
<n:getRateResponse xmlns:n='urn:xmethods-CurrencyExchange'>
<Result xsi:type='xsd:float'>0.7206</Result>
</n:getRateResponse>
</soap:Body>

</soap:Envelope>
```

Again, some whitespace has been inserted to make things a bit easier to understand. This is a garden-variety HTTP response, similar to those discussed in Chapter 3. Here, the body is another XML document, a somewhat shorter one, that contains a Result element. This is the single value that was returned by the call to getRate(). It is a float value: 0.7206, which is the ratio of one U.S. dollar to one Australian dollar at the time of the call to the Web service.

All of these traces were done with Ethereal (http://www.ethereal.com/), a free and open-source network analyzer. Figure 4.1 shows Ethereal in action.

![Ethereal screenshot](image)

**Figure 4.1:** Ethereal makes it easy to monitor SOAP activity.
4.2 Implementing SOAP in PHP

Three cheers for Dietrich Ayala, who has done the PHP community a favor by creating NuSOAP. NuSOAP is a PHP library—a series of PHP files you can easily import into your own programs by means of a require or require_once statement—that supports the creation of SOAP clients and servers. It's available at http://dietrich.ganx4.com/nusoap/index.php.

It's blisteringly easy to create a SOAP server object with NuSOAP, then register your own functions with it. It's that easy to convert your functions to Web services. Invoking them is similarly simple. In essence, there are four steps involved in exposing a function as a Web service with NuSOAP:

1. Import the NuSOAP library,
2. Instantiate a soap_server object,
3. Register a function with the soap_server object using the register() function, and
4. Direct the incoming HTTP data to the soap_server object using the service() function.

Section 4.2.1 shows you how to carry out the four steps in more detail.

There are a couple of other SOAP implementations in progress, and in various stages of readiness. Among them are the PHP-SOAP, in alpha testing at the time of this writing (http://phpsoaptoolkit.sourceforge.net/phpsoap/) and the SOAP module in PHP Extension and Application Repository (PEAR), in beta (http://pear.php.net/package/SOAP).

4.2.1 A Simple Application of NuSOAP

Let's have a look at a simple application of NuSOAP.

Setting Up a SOAP Server

To establish a SOAP server with NuSOAP, you must first have a function you wish to expose as a Web service. Any function is suitable for the purpose. In this example, we'll use this one:

```php
function hello($name) {
    return "hello $name!";
}
```

That's a very simple function that does nothing but take a string as a parameter, insert that string into a larger string, and return the result. Notice that there's nothing unusual about this function, which is going to be exposed as a Web service.
To expose the function as a Web service, we must first import the NuSOAP library:

```php
require_once('nusoap-0.6/nusoap.php');
```

Then, we must create a soap_server object (soap_server objects are defined in nusoap.php)

```php
$s = new soap_server;
```

and register our function with the soap_server object:

```php
$s->register('hello');
```

This last line is interesting. It invokes the register() method of the object known as $s, which in this case is the soap_server object. The parameter passed to register() is the name of the function we wish to expose as a Web service, namely hello(), but without the customary parentheses.

The only other characteristic the server program needs is a line of code that refers incoming HTTP POST requests to the NuSOAP library for evaluation. Again, the solution is to invoke a method of $s, the soap_server object:

```php
$s->service($HTTP_RAW_POST_DATA);
```

It's a black box, after all. Incoming HTTP POST requests get sent to the soap_server object, where they're passed on to one of the registered functions as appropriate. By the way, $HTTP_RAW_POST_DATA is a special variable that the PHP interpreter makes available to programs whenever the incoming data is of a Multimedia Internet Mail Extension (MIME) type other than application/x-www.form-urlencoded, which is the MIME type that's usually associated with a form. In the case of a SOAP request, the MIME type is application/soap+xml, so the contents of the HTTP POST request are accessed through $HTTP_RAW_POST_DATA.

What, then, sends the HTTP POST request? The SOAP client does.

### Setting Up a SOAP Client

There's no point in having a Web service, otherwise known as a SOAP server, if there isn't also something to call it. NuSOAP makes SOAP clients even easier to create than SOAP servers.

NuSOAP called its SOAP client object a soapclient (in contrast to soap_server), so be careful with your underscore characters. When you create a soapclient object, you have to associate it with the server to which it will refer. This reference takes the form of a full HTTP uniform resource locator (URL). The URL specifies the location of the PHP containing the Web services (functions and soap_server object) that the client will access.

To create the soapclient object, you must first invoke the NuSOAP library, just as was required on the server

```php
require_once('nusoap-0.6/nusoap.php');
```
then instantiate a soapclient object with the full URL of the server file as a parameter:

```php
```

With the client object in place, it is possible to invoke a remote function through the `call()` function of the soapclient object. The `call()` function takes the name of the remote function and an array containing the parameters to be sent to that remote function, so the first step is to populate the array of parameter. The names for the elements in the array correspond to the name of the arguments specified in the functions that are exposed as Web services:

```php
$parameters = array('name'=>'David');
```

With that array established, we can invoke the `call()` function of the soapclient object, known in this case by the handle `$client`:

```php
echo $client->call('hello',$parameters);
```

That line sends the `$parameters` array to the `hello()` function on the SOAP server associated with `$client`, and the result is written to output.

One could not reasonably ask for a simpler means of making SOAP calls across an HTTP link. NuSOAP enables us to import a library, then expose PHP functions as Web services with just a couple of lines of code. Referring to those services as a client is even easier.

### 4.2.2 A More Complex Application of NuSOAP

Here, we'll put together a simple program that makes the call to the XMethods.net `getRate()` Web service that was documented in the beginning of this chapter. It's a simple program that requires only a special file written in Web services description language (WSDL) to send the required call to the remote resource.

Here's a full listing of the program that makes the call.

```
demoSOAP.php
```

```php
require_once('nusoap-0.6/nusoap.php'); // Import NuSOAP library.

$countryl = $_GET['countryl']; // Get user input from HTML form.
$countryle = $_GET['countryle']; // Get user input from HTML form.

$parameters = array(
    'countryl' => $countryl,
    'countryle' => $countryle
);```
4.3 Questions and Exercises

1. Consider the security implications of using SOAP. To what extent can SOAP be made secure by using Secure Sockets Layer (SSL) encryption to carry HTTP—HTTPS Secure (HTTPS)—instead of plain HTTP? How about by running SOAP on ordinary HTTP, but over a virtual private network?
2. Write a simple application that, instead of responding with a simple datatype, sends back an associative array. Can you do one that sends back an object?

3. What are some reasonable applications for Web services on the public Internet? What are some applications that might not work well for technical reasons?

4. Do some research into standards documents and explain the relationship between SOAP and XML-Remote Procedure Call (XML-RPC) in terms of Web services.