12 1690 CH09 4/25/02 2:01 PM Page 209

9 PHP and Web Services

WEB SERVICES ARE ONE OF THE MOST talked-about technologies of the day. They are set to change how data is exchanged on the Internet as the Internet itself evolves to deliver content not only to web browsers on PCs but also to PDAs and other devices. Further still, the evolution of "Internet-ready" software and hardware will see web services being used in applications such as MP3 players, personal stereos, and game consoles. PHP is a great language for developing web services, and this chapter shows you just what a web service is and how it is made up. Then we will look at how you can use PHP to develop web services.

What Makes Up a Web Service?

A web service is made up of four parts, as shown in Figure 9.1. The first part is the component that wants to act as a web service. It can be any part of an application: the executable, a COM component, a JavaBean, and so on. The web service component exposes public methods and functions that other applications can query. Your component could be one that you have created for this purpose, but normally you can expose any component that has public methods as a web service.

12 1690 CH09 4/25/02 2:01 PM Page 210

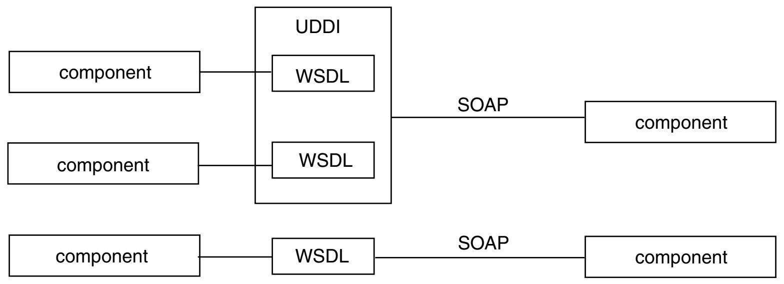


Figure 9.1 The makeup of web services.

You must allow your components to be accessed by other applications as a web service. To do this, you must allow other applications to see what public functions and methods your components have. You don't allow applications to do this directly. To accomplish this, you create a file based on an XML-based metalanguage called Web Services Description Language (WSDL).

Next you must allow applications to query the WSDL file and exchange data with the web service.

For this purpose, you use another XML-based metalanguage called Simple Object Access Protocol (SOAP).

Using SOAP, you look up the web service component's public methods and functions using the WSDL file and then query those methods and functions. However, you don't always know where to find a web service's WSDL file. In that case, you can look at an XML-based database (also called a *registry*) of WSDL addresses. This database is called a Uniform Description, Discovery, and Integration (UDDI) registry.

The easiest way to remember the component parts of a web service is to think in terms of these three concepts:

- Discovery: UDDI lets you discover web services.
- Query: WSDL lets you query web services.
- Transport: SOAP lets you transmit those queries back and forth from the web service.

Don't worry if some of these terms are new to you. They are covered in further detail later in this chapter. Now that we have identified all the component parts of a web service, let's look at each in detail.

SOAP (Simple Object Access Protocol)

In Chapter 5, "PHP and Sessions," SOAP was mentioned briefly when we looked at how WDDX developed. SOAP, like WDDX, is an XML-based language. Unlike WDDX, however, and like another XML-based language,

12 1690 CH09 4/25/02 2:01 PM Page 211

XML-RPC, it is used for RPC (Remote Procedure Call) via XML. XML-RPC started life as an idea of Dave Winer of UserLand software. He discussed his ideas with Microsoft, and from his ideas, SOAP was born. (XML-RPC continues to develop as a protocol separate from SOAP.)

Other companies (such as IBM) joined Microsoft in developing SOAP. Soon after that, implementations of SOAP for languages such as Java (via IBM) and Visual C++ and Visual Basic (via Microsoft) were released.

Using SOAP

So how is SOAP used? SOAP is an XML-based language, so in effect, all SOAP implementations do is create XML files or strings that facilitate passing data and calling methods between (normally remote) applications. SOAP does not have to be strictly about web services; in other words, it does not require WSDL or UDDI to work and can be used in a standard RPC manner. SOAP is often described as passing objects between applications; this can be both misleading and confusing. All SOAP does is allow a public function or method to be queried via an XML interface. It does not pass physical objects in the same way that Java serialization does, for example.

When an application queries a public function or another application's methods, it passes some data to that function or method and might get results in return. A public function or method does not always do this, but it is good practice that such functions or methods at least return some handshake data (a simple code that allows the calling application to see that the public function or method has received the data it sent).

When dealing with remote applications, you might face data type problems. That is, different languages have different ways of representing data. You might have some success on this front. For example, Java and PHP have very similar data types, but when dealing with calling applications that might be made up of languages such Perl, Python, and C++, you will face a nightmare.

If you imagine that your remote application is developed in Java and your calling clients are made up of PHP and Visual Basic, you might face few problems with the PHP application calling the remote Java application. But you will face a lot of problems when you do the same with a Visual Basic application.

Luckily, SOAP deals with this by presenting a standard way of representing data. Imagine that your remote Java application function accepts a string. Using SOAP, you translate the calling application data into the SOAP equivalent and pass that to the remote Java application. The Java application then translates the SOAP string data into the Java equivalent.

This can work in reverse too so that if your Java application returns a result, it does so as SOAP data for your client applications to translate back into

12 1690 CH09 4/25/02 2:01 PM Page 212

native data types. So, in effect, SOAP does not really let you pass objects between applications. Instead, it provides the means to interface between different objects in different languages.

I was once asked if SOAP, based on part of its definition (simple object), is for simple objects only. Its meaning is not to be confused with objects in the sense we use them in OOP languages such as C++. It does not need to know what an object does, how it exists, or how it works. In fact, the very use of the word *object* can be misleading. SOAP does not require objects to exist in a true OOP sense. They can be nothing more than public functions and methods and are not subject to OOP concepts such as encapsulation.

SOAP Transparency

Because SOAP is XML-based, it is nothing more than ASCII data that is being transmitted (and therefore is as simple as standard text). This is one of key benefits of SOAP. It uses simple character encoding (ASCII) as a file format. It can be transmitted on any protocol that supports the transmission of ASCII data. As it happens, most TCP/IP protocols do. This allows SOAP to be used across HTTP (the most common protocol to be used with SOAP), FTP, SMTP, and so on. This brings an added benefit in that such protocols don't require special ports and security measures such as firewalls. They run through commonly used ports. (This advantage is also enjoyed by other XML-based RPC methods, such as XML-RPC.)

This property is also apparent when you compare SOAP to other RPC methods. In PHP, you can also make use of DCOM (which is used with COM), CORBA (which uses the IIOP protocol), and Java (which natively supports the RMI protocol). Protocols that facilitate RPC are called *wire protocols* because they are low-level and require special ports. SOAP, however, has no such requirements.

SOAP's Makeup

What exactly is SOAP made of? A SOAP message is called a SOAP *envelope*. The following code exemplifies a SOAP envelope:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle=
➥"http://schemas.xmlsoap.org/soap/encoding/"
➥ xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

</SOAP-ENV:Envelope>
```

An envelope contains a body, which can be either a SOAP body call or a SOAP body response.

SOAP Body Call

When a call is made to a public function or method, this is done with a SOAP body call. Such a body call looks something like this:

```
<SOAP-ENV:Body>
<SOAPSDK1:HelloFunc xmlns:SOAPSDK1="http://tempuri.org/message/">
<uname xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
➥ xmlns:SOAPSDK3="http://www.w3.org/2001/XMLSchema"
➥ SOAPSDK2:type="SOAPSDK3:string">Andrew</uname>
</SOAPSDK1:HelloFunc>
</SOAP-ENV:Body>
```

Here the `HelloFunc` method is passed a string of data called `"Andrew"`. Note that SOAP has added a mapping of what data we are passing to the public service or method:

```
type="SOAPSDK3:string"
```

A completed SOAP envelope calling a public service or method looks like the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle=
➥"http://schemas.xmlsoap.org/soap/encoding/"
➥ xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<SOAPSDK1:HelloFunc xmlns:SOAPSDK1="http://tempuri.org/message/">
<uname xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
➥ xmlns:SOAPSDK3="http://www.w3.org/2001/XMLSchema"
➥ SOAPSDK2:type="SOAPSDK3:string">Andrew</uname>
</SOAPSDK1:HelloFunc>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Body Response

In response to a call to a public function or method, SOAP can respond to that call using the SOAP body response:

```
<SOAP-ENV:Body>
<SOAPSDK1:HelloFuncResponse xmlns:SOAPSDK1="http://tempuri.org/message/">
<Result xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
➥ xmlns:SOAPSDK3="http://www.w3.org/2001/XMLSchema"
➥ SOAPSDK2:type="SOAPSDK3:string">hello Andrew</Result>
<uname xmlns:SOAPSDK4="http://www.w3.org/2001/XMLSchema-instance"
➥ xmlns:SOAPSDK5="http://www.w3.org/2001/XMLSchema"
➥ SOAPSDK4:type="SOAPSDK5:string">Andrew</uname>
</SOAPSDK1:HelloFuncResponse>#
</SOAP-ENV:Body>
```

12 1690 CH09 4/25/02 2:01 PM Page 214

The SOAP body result contains any data that the public function or method returns:

```
<Result xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
➥ xmlns:SOAPSDK3="http://www.w3.org/2001/XMLSchema"
➥ SOAPSDK2:type="SOAPSDK3:string">hello Andrew</Result>
```

along with the original method call:

```
<uname xmlns:SOAPSDK4="http://www.w3.org/2001/XMLSchema-instance"
➥ xmlns:SOAPSDK5="http://www.w3.org/2001/XMLSchema"
➥ SOAPSDK4:type="SOAPSDK5:string">Andrew</uname>
```

Web Services Description Language (WSDL)

In the web services sense, although SOAP helps you exchange data between the public functions or methods of a web service, it can't help you explain which public functions or methods are available and what they do. Without this information, you can't use SOAP to exchange data, because you have no idea what is available to help you facilitate the exchange.

For this purpose, we have Web Services Description Language (WSDL). Like SOAP, WSDL is an XML-based file format for describing what public functions and methods are available in a web service. Other applications use the WSDL file to find this information and then use SOAP against those described public functions or methods.

WSDL File Makeup

```
<?xml version='1.0' encoding='UTF-8' ?>
 <!-- Generated 09/24/01 by Microsoft SOAP Toolkit WSDL File Generator,
➥ Version 1.02.813.0 -->
<definitions  name ='PHP4WINSOAP'
➥   targetNamespace = 'http://tempuri.org/wsdl/'
      xmlns:wsdlns='http://tempuri.org/wsdl/'
      xmlns:typens='http://tempuri.org/type'
      xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
      xmlns:xsd='http://www.w3.org/2001/XMLSchema'
      xmlns:stk='http://schemas.microsoft.com/soap-toolkit/wsdl-extension'
      xmlns='http://schemas.xmlsoap.org/wsdl/'>
  <types>
    <schema targetNamespace='http://tempuri.org/type'
      xmlns='http://www.w3.org/2001/XMLSchema'
      xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
      xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
      elementFormDefault='qualified'>
    </schema>
  </types>
  <message name='Examples.HelloFunc'>
```

```
      <part name='uname' type='xsd:anyType'/>
    </message>
    <message name='Examples.HelloFuncResponse'>
      <part name='Result' type='xsd:anyType'/>
      <part name='uname' type='xsd:anyType'/>
    </message>
    <portType name='ExamplesSoapPort'>
      <operation name='HelloFunc' parameterOrder='uname'>
        <input message='wsdlns:Examples.HelloFunc' />
        <output message='wsdlns:Examples.HelloFuncResponse' />
      </operation>
    </portType>
    <binding name='ExamplesSoapBinding' type='wsdlns:ExamplesSoapPort' >
      <stk:binding preferredEncoding='UTF-8'/>
      <soap:binding style='rpc' transport=
      ➥'http://schemas.xmlsoap.org/soap/http' />
      <operation name='HelloFunc' >
        <soap:operation soapAction=
        ➥'http://tempuri.org/action/Examples.HelloFunc' />
        <input>
          <soap:body use='encoded' namespace='http://tempuri.org/message/'
              encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
        </input>
        <output>
          <soap:body use='encoded' namespace='http://tempuri.org/message/'
              encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
        </output>
      </operation>
    </binding>
    <service name='PHP4WINSOAP' >
      <port name='ExamplesSoapPort' binding='wsdlns:ExamplesSoapBinding' >
        <soap:address location='http://localhost/phpbook/Chapter9
        ➥_SOAP/SOAP/Server/PHP4WINSOAP.ASP' />
      </port>
    </service>
  </definitions>
```

As you can see, the WSDL file format is quite a complicated one. The three most important pieces are the port type, binding, and service name.

Port Type

The port type defines which public functions or methods are available. It uses the full name, which in this case is the `HelloFunc` method of the `Examples` class:

```
<portType name='ExamplesSoapPort'>
  <operation name='HelloFunc' parameterOrder='uname'>
    <input message='wsdlns:Examples.HelloFunc' />
    <output message='wsdlns:Examples.HelloFuncResponse' />
  </operation>
</portType>
```

12 1690 CH09 4/25/02 2:01 PM Page 216

Binding

Binding describes how calls are made to each public method or function of a web service. In other words, it describes what encoding a client application should use when querying that public method or function.

```
<binding name='ExamplesSoapBinding' type='wsdlns:ExamplesSoapPort' >
    <stk:binding preferredEncoding='UTF-8'/>
    <soap:binding style='rpc'
    ➥ transport='http://schemas.xmlsoap.org/soap/http' />
    <operation name='HelloFunc' >
      <soap:operation soapAction=
      ➥'http://tempuri.org/action/Examples.HelloFunc' />
      <input>
        <soap:body use='encoded' namespace='http://tempuri.org/message/'
            encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
      </input>
      <output>
        <soap:body use='encoded' namespace='http://tempuri.org/message/'
            encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
      </output>
    </operation>
  </binding>
```

This line describes what encoding will use of the binding:

```
<stk:binding preferredEncoding='UTF-8'/>
```

This must match the encoding for your XML file (the encoding specified in the XML header). Next you specify what the binding is (RPC) and its transport (HTTP):

```
<soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />
```

Next you specify what function you will call as `Hello Func`:

```
<operation name='HelloFunc' >
<soap:operation soapAction='http://tempuri.org/action/Examples.HelloFunc' />
```

The WSDL file describes the encoding you use to query the web service and obtain a result:

```
<input>
<soap:body use='encoded' namespace='http://tempuri.org/message/'
➥ encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
</input>
<output>
<soap:body use='encoded' namespace='http://tempuri.org/message/'
➥ encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
</output>
```

Although it's important to understand the structure of a WSDL file, you won't often need to write a WSDL file yourself. Most web service toolkits have tools for creating WSDL files for you.

Service Name

The service name defines where the WSDL gateway will be. A WSDL gateway is used to query the web service's component. A WSDL file passes the gateway information back to the client application. Such queries from the client application go to the gateway and on to the web service component and back.

```
<service name='PHP4WINSOAP' >
  <port name='ExamplesSoapPort' binding='wsdlns:ExamplesSoapBinding' >
    <soap:address location='http://localhost/phpbook/Chapter9
    ➥_SOAP/SOAP/Server/PHP4WINSOAP.ASP' />
  </port>
</service>
```

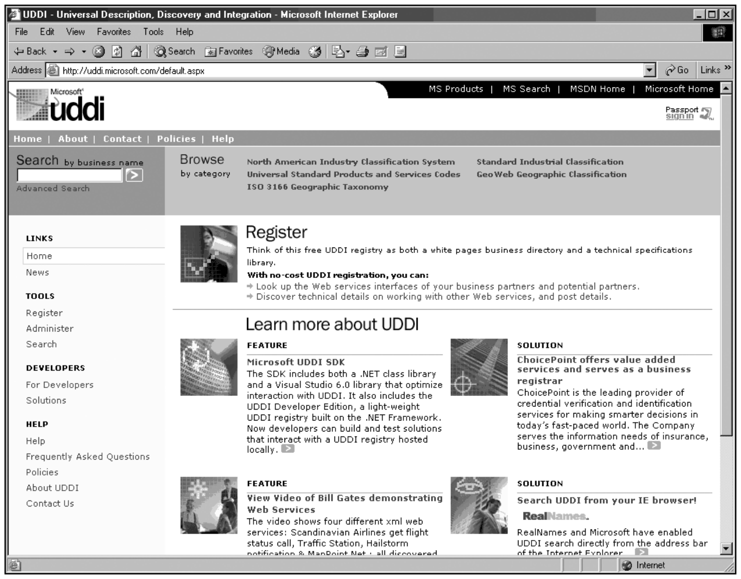
A gateway is language-independent. For instance, our example uses ASP, but the gateway can be developed in any language you like, as long as it supports XML. All it does is work in unison with the WSDL file to allow client applications to pass SOAP queries back and forth between the web service and the client application.

Uniform Description, Discovery, and Integration (UDDI)

Although you can now look up and query a web service's public methods and functions, finding a web service presents a problem. It can't be found using normal means such as a web search engine. If your company wants to make use of web services, it needs to make them generally available to all calling applications, not simply provide URLs on web pages for people to connect to.

Web services need a source of information all their own that an application can search and use to find WSDL files to query. UDDI was created for this purpose (see `http://www.uddi.org`). UDDI is another emerging standard for web services. It allows a company to register its company details, web page, and so on with a business group (for example, a car dealership could register with an automotive association). A person can then search the UDDI for companies within a business group or type via an application or directly via a web browser. More interestingly, UDDI allows companies to publish details of what web services are available (using brief descriptions and keywords) as well as the URLs of the WSDL files. An application can use this information to discover which web services a company has available.

Several public UDDI registries have been set up to allow companies to test and publish details of real-world web services. The Microsoft UDDI registry (`http://uddi.microsoft.com`), shown in Figure 9.2, serves such a purpose.


UDDI - Universal Description, Discovery and Integration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://uddi.microsoft.com/default.aspx
Go Links
MS Products | MS Search | MSDN Home | Microsoft Home
Microsoft uddi
Passport sign in
Home | About | Contact | Policies | Help
Search by business name
Advanced Search
Browse by category
North American Industry Classification System
Universal Standard Products and Services Codes
ISO 3166 Geographic Taxonomy
Standard Industrial Classification
GeoWeb Geographic Classification
LINKS
Home
News
TOOLS
Register
Administer
Search
DEVELOPERS
For Developers
Solutions
HELP
Help
Frequently Asked Questions
Policies
About UDDI
Contact Us
Internet

Register

Think of this free UDDI registry as both a white pages business directory and a technical specifications library.

With no-cost UDDI registration, you can:

- Look up the Web services interfaces of your business partners and potential partners.
- Discover technical details on working with other Web services, and post details.

Learn more about UDDI

FEATURE

Microsoft UDDI SDK

The SDK includes both a .NET class library and a Visual Studio 6.0 library that optimize interaction with UDDI. It also includes the UDDI Developer Edition, a light-weight UDDI registry built on the .NET Framework. Now developers can build and test solutions that interact with a UDDI registry hosted locally.

SOLUTION

ChoicePoint offers value added services and serves as a business registrar

ChoicePoint is the leading provider of credential verification and identification services for making smarter decisions in today's fast-paced world. The Company serves the information needs of insurance, business, government and...

FEATURE

View Video of Bill Gates demonstrating Web Services

The video shows four different xml web services: Scandinavian Airlines get flight status call, Traffic Station, Hailstorm notification & MapPoint.Net : all discovered

SOLUTION

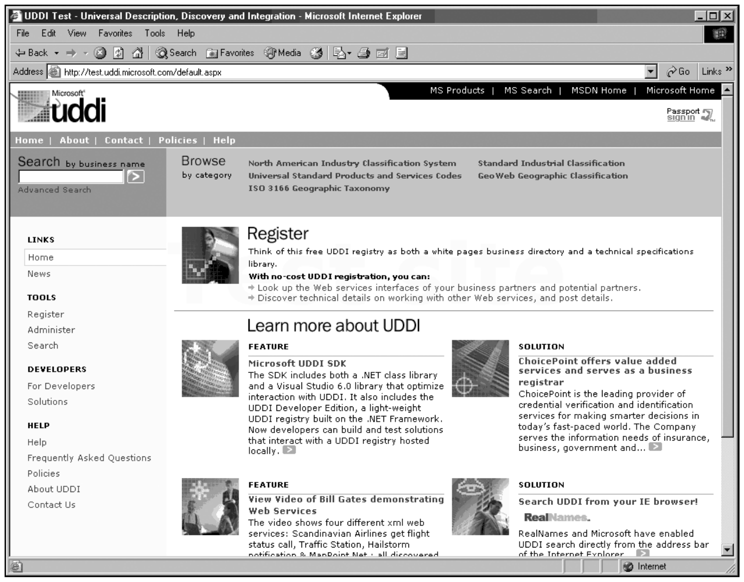
Search UDDI from your IE browser!

RealNames.

RealNames and Microsoft have enabled UDDI search directly from the address bar of the Internet Explorer

Figure 9.2 Microsoft's UDDI web site.

Microsoft also provides a test UDDI registry at `http://test.uddi.microsoft.com` (see Figure 9.3). Other UDDI registries are available from IBM at `http://www.ibm.com/developerworks/webservices/`.


UDDI Test - Universal Description, Discovery and Integration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://test.uddi.microsoft.com/default.aspx
Go Links
MS Products | MS Search | MSDN Home | Microsoft Home
Microsoft uddi
Passport sign in
Home | About | Contact | Policies | Help
Search by business name
Advanced Search
Browse by category
North American Industry Classification System
Universal Standard Products and Services Codes
ISO 3166 Geographic Taxonomy
Standard Industrial Classification
GeoWeb Geographic Classification
LINKS
Home
News
TOOLS
Register
Administer
Search
DEVELOPERS
For Developers
Solutions
HELP
Help
Frequently Asked Questions
Policies
About UDDI
Contact Us
Internet

Register

Think of this free UDDI registry as both a white pages business directory and a technical specifications library.

With no-cost UDDI registration, you can:

- Look up the Web services interfaces of your business partners and potential partners.
- Discover technical details on working with other Web services, and post details.

Learn more about UDDI

FEATURE

Microsoft UDDI SDK

The SDK includes both a .NET class library and a Visual Studio 6.0 library that optimize interaction with UDDI. It also includes the UDDI Developer Edition, a light-weight UDDI registry built on the .NET Framework. Now developers can build and test solutions that interact with a UDDI registry hosted locally.

SOLUTION

ChoicePoint offers value added services and serves as a business registrar

ChoicePoint is the leading provider of credential verification and identification services for making smarter decisions in today's fast-paced world. The Company serves the information needs of insurance, business, government and...

FEATURE

View Video of Bill Gates demonstrating Web Services

The video shows four different xml web services: Scandinavian Airlines get flight status call, Traffic Station, Hailstorm notification & MapPoint.Net : all discovered

SOLUTION

Search UDDI from your IE browser!

RealNames.

RealNames and Microsoft have enabled UDDI search directly from the address bar of the Internet Explorer

Figure 9.3 Microsoft's UDDI web site for testing.

12 1690 CH09 4/25/02 2:01 PM Page 219

The Future of Web Services

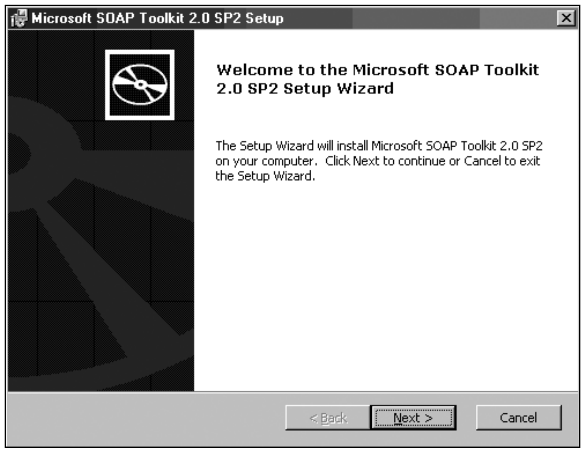
In May 2000, SOAP was submitted to the W3C for standardization. (That group included Microsoft, UserLand, DeveloperMentor, and IBM. The submission can be found at `http://www.w3.org/Submission/2000/05/`.) In time, SOAP will become part of a brand-new protocol that the W3C is calling XML Protocol. (It's called XP for short, but don't confuse it with the Microsoft operating system of the same name.) XP will serve exactly the same purpose as SOAP but will be the first standardized XML-based RPC protocol. The drafting of XP is under way, with the W3C XP working group releasing a draft of the SOAP 1.2 specification. Further details on XP can be found at `http://www.w3.org/2000/xp/`.

Using PHP to Create Web Services

Now that we have looked at what a web service is and what comprises it, let's now look at how you can create a web service in PHP.

Creating the Web Service

To create your web service, you will use the Microsoft SOAP SDK (`http://msdn.microsoft.com/webservices/`). It gives you all you need to create web services and web service clients (connecting to and querying web services). The client portion of the SDK is included in the Windows XP operating system. The SDK is continually updated by Microsoft as the various protocols (such as SOAP and WSDL) change. The SDK used in this chapter is the Microsoft SOAP SDK 2.0 SP2 (see Figure 9.4).


Microsoft SOAP Toolkit 2.0 SP2 Setup

Welcome to the Microsoft SOAP Toolkit 2.0 SP2 Setup Wizard

The Setup Wizard will install Microsoft SOAP Toolkit 2.0 SP2 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Figure 9.4 The Microsoft SOAP SDK installer dialog box.

Other SOAP SDKs are also available, such as the IBM web service SDK, which includes web service libraries and a WSDL file generator (`http://www-106.ibm.com/developerworks/webservices/`). Both IBM and Microsoft provide SDKs for UDDI that are separate from their SOAP/WSDL SDKs.

Creating a Web Service Component

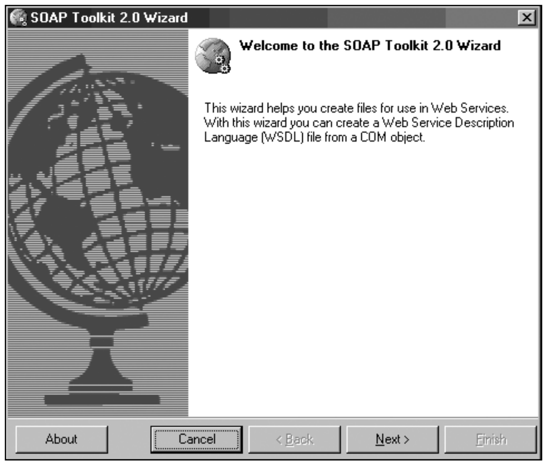
The base of your web service (the one that provides the public functions or methods that you want to query against) is the simple COM component you developed in Chapter 7, "PHP, COM, and .NET." You developed this in Visual Basic using the following code:

```
Option Explicit
Public Function HelloFunc(ByRef uname As Variant) As Variant
     HelloFunc = "hello " & uname
End Function
```

All this COM component does is take a string as its argument (such as `"everyone"`) and return a string (such as `"hello everyone"`). You can either compile a new version of this COM component for use with your web service or simply reuse the same COM component exactly as you left it in Chapter 7.

Creating a WSDL File

As soon as you have your COM component, you need to make a WSDL file to turn it into a web service. Although you can write this by hand, as you have seen, this is quite complicated. Luckily, the Microsoft SOAP SDK provides you with a handy WSDL generation tool called wsdlgen.exe. (You can access it by selecting Start, SOAP SDK.) When you start the WSDL generation tool, you see a welcome screen, as shown in Figure 9.5.



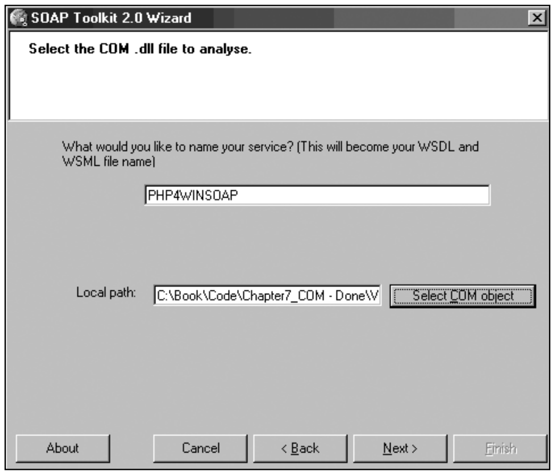
Welcome to the SOAP Toolkit 2.0 Wizard

This wizard helps you create files for use in Web Services. With this wizard you can create a Web Service Description Language (WSDL) file from a COM object.

Figure 9.5 The Microsoft WSDL Wizard welcome dialog box.

12 1690 CH09 4/25/02 2:01 PM Page 221

If you click Next, you can select which COM component you want to create the WSDL file for (its physical file location), as shown in Figure 9.6. Also select a name for your WSDL. You can use any name, but it is best to choose something short and easy to use.



Select the COM .dll file to analyse.

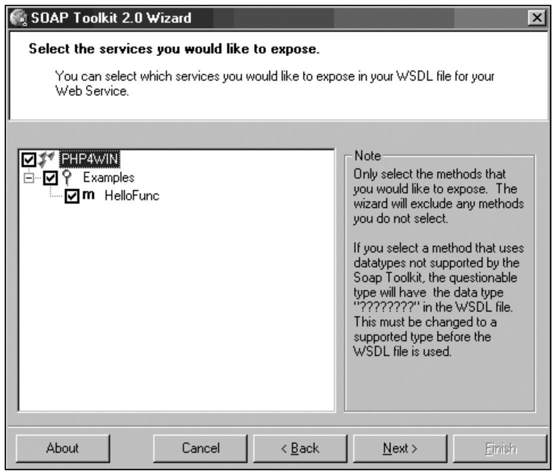
What would you like to name your service? (This will become your WSDL and WSML file name)

PHP4WINSOAP

Local path: C:\Book\Code\Chapter7_COM - Done\V

Figure 9.6 The Microsoft WSDL Wizard COM object selection dialog box.

When you click Next, the tool checks what public functions and methods your COM component has available and allows you to select which ones you want to expose in the WSDL file, as shown in Figure 9.7. You might not want to expose them all, but you can do so if you want to.



Select the services you would like to expose.

You can select which services you would like to expose in your WSDL file for your Web Service.

- ☑ PHP4WIN
  - ☑ Examples
    - ☑ m HelloFunc

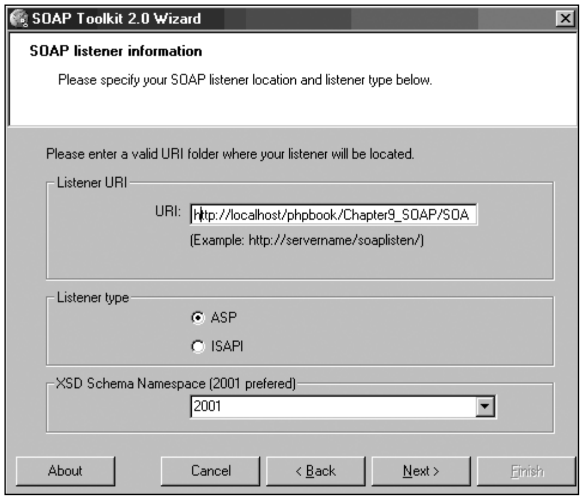
Only select the methods that you would like to expose. The wizard will exclude any methods you do not select.

If you select a method that uses datatypes not supported by the Soap Toolkit, the questionable type will have the data type "????????" in the WSDL file. This must be changed to a supported type before the WSDL file is used.

Figure 9.7 The Microsoft WSDL Wizard function selection dialog box.

12 1690 CH09 4/25/02 2:01 PM Page 222

When you click Next, you reach the SOAP listener information dialog box, shown in Figure 9.8. This is quite an important section, because this is where you map the WSDL file to your COM component. This tool lets you create either an ASP listener or an ISAPI listener (also called a gateway). The listener can be stored in either the same place as your COM component or elsewhere. Make sure, however, that the directory you choose is accessible by the web server.


SOAP Toolkit 2.0 Wizard

SOAP listener information

Please specify your SOAP listener location and listener type below.

Please enter a valid URI folder where your listener will be located.

Listener URI

URI: http://localhost/phpbook/Chapter9_SOAP/SOA

(Example: http://servername/soaplisten/)

Listener type

[✓ ASP]
ISAPI

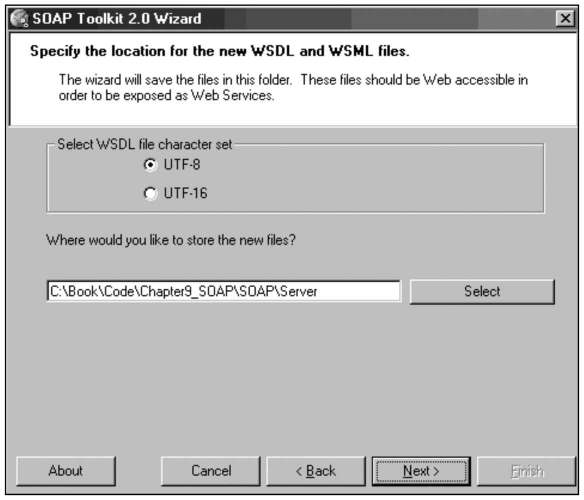
XSD Schema Namespace (2001 prefered)

2001

Figure 9.8 The Microsoft WSDL Wizard listener setup dialog box.

After you click Next, you select where you want to store the tool you will create (see Figure 9.9). The WSDL file can be separate from the COM component if you want.

12 1690 CH09 4/25/02 2:01 PM Page 223



SOAP Toolkit 2.0 Wizard

Specify the location for the new WSDL and WSML files.

The wizard will save the files in this folder. These files should be Web accessible in order to be exposed as Web Services.

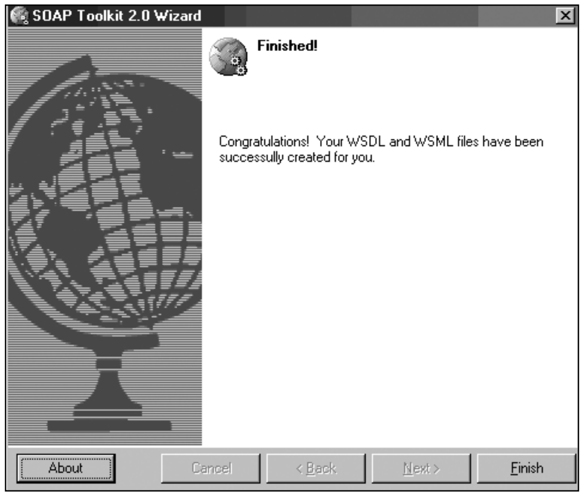
Select WSDL file character set
[✓ UTF-8]
UTF-16

Where would you like to store the new files?

C:\Book\Code\Chapter9_SOAP\SOAP\Server

Figure 9.9 The Microsoft WSDL Wizard file storage dialog box.

Click Next. The WSDL file is created (see Figure 9.10).



SOAP Toolkit 2.0 Wizard

Finished!

Congratulations! Your WSDL and WSML files have been successfully created for you.

Figure 9.10 The Microsoft WSDL Wizard completion dialog box.

12 1690 CH09 4/25/02 2:01 PM Page 224

Client Application

To make use of the services your web service provides, you must access the WSDL file. Your client application does just this. You can use two methods in PHP to create the client application. You can use the COM objects that the Microsoft SOAP SDK provides, or you can use a native PHP implementation.

Using the Microsoft SOAP SDK COM Objects

When using the COM objects in the SDK, you can approach creating the client application using two methods. You can use the COM objects directly from PHP, or you can wrap them into a single COM object.

Using COM Objects Directly from PHP

The SOAP SDK provides you with a set of COM objects for querying a web service, as follows:

```
<?php

//load COM SOAP client object
$soapob = new COM("MSSOAP.SoapClient");

//connect to web service
$soapob->mssoapinit("http://localhost/phpbook/Chapter9
➥_SOAP/SOAP/Server/PHP4WINSOAP.WSDL");

//obtain result from web service method
$soapmessage = $soapob->HelloFunc("Andrew");

//print result
print($soapmessage);

?>
```

First, you load the SOAP client COM object into memory:

```
$soapob = new COM("MSSOAP.SoapClient");
```

Next you connect to your web service. (Remember to use the full URL of where you stored the WSDL file you created earlier in this chapter.)

```
$soapob->mssoapinit("http://localhost/phpbook/Chapter9
➥_SOAP/SOAP/Server/PHP4WINSOAP.WSDL");
```

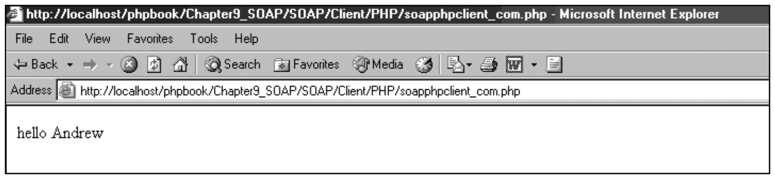
Next you call the `HelloFunc` method of your web service, passing the string `"Andrew"` and storing its return result:

```
$soapmessage = $soapob->HelloFunc("Andrew");
```

Finally, you display the result:

```
print($soapmessage);
```

If you run the PHP script, you should see the result from the `HelloFunc` method of your web service, as shown in Figure 9.11.


http://localhost/phpbook/Chapter9_SOAP/SOAP/Client/PHP/soapphpclient_com.php - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/phpbook/Chapter9_SOAP/SOAP/Client/PHP/soapphpclient_com.php

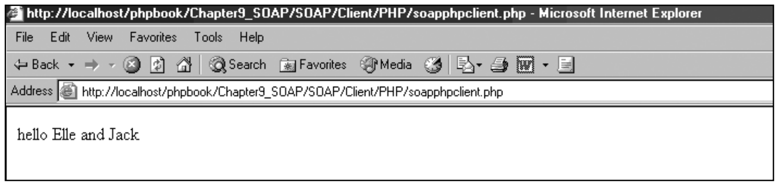
```
hello Andrew
```

Figure 9.11 Your web service displaying data.

If you change the call to the web service method, the output changes. For example, if you change the following line in your script:

```
$soapmessage = $soapob->HelloFunc("Elle and Jack");
```

the output changes, as shown in Figure 9.12.


http://localhost/phpbook/Chapter9_SOAP/SOAP/Client/PHP/soapphpclient.php - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/phpbook/Chapter9_SOAP/SOAP/Client/PHP/soapphpclient.php

```
hello Elle and Jack
```

Figure 9.12 Your web service displaying different data.

Your Web Service Wrapped into a Single COM Object

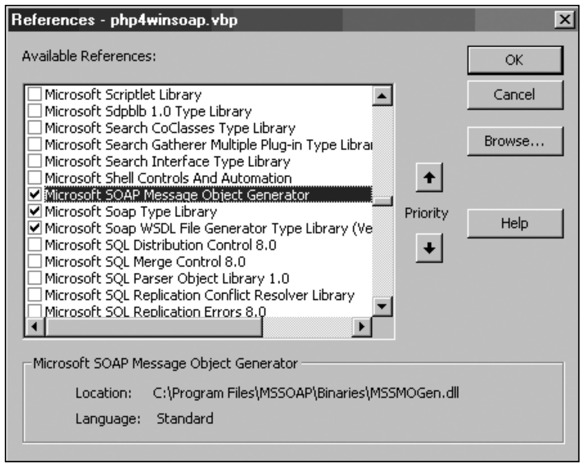
You might want to reuse the same SOAP client application code across different scripts. In such a case, you can wrap the client application into a COM object.

If you start Visual Basic and create an ActiveX DLL called php4winsoap with a class called `Output`, you can add the following code:

```
Public Function getdata()

Set sc = New SoapClient

On Error Resume Next

sc.mssoapinit "http://localhost/phpbook/Chapter9
➥_SOAP/SOAP/Server/PHP4WINSOAP.WSDL"

If Err <> 0 Then
```

```
  getdata = "initialization failed " + Err.Description

End If

getdata = sc.HelloFunc("Andrew")


End Function
```

If you try to compile this code, you will get an error. You must also reference the SOAP COM objects in your visual basic project, as shown in Figure 9.13.



- [ ] Microsoft Scriptlet Library
- [ ] Microsoft Sdpblb 1.0 Type Library
- [ ] Microsoft Search CoClasses Type Library
- [ ] Microsoft Search Gatherer Multiple Plug-in Type Librar
- [ ] Microsoft Search Interface Type Library
- [ ] Microsoft Shell Controls And Automation
- [x] Microsoft SOAP Message Object Generator
- [x] Microsoft Soap Type Library
- [x] Microsoft Soap WSDL File Generator Type Library (Ve
- [ ] Microsoft SQL Distribution Control 8.0
- [ ] Microsoft SQL Merge Control 8.0
- [ ] Microsoft SQL Parser Object Library 1.0
- [ ] Microsoft SQL Replication Conflict Resolver Library
- [ ] Microsoft SQL Replication Errors 8.0

Microsoft SOAP Message Object Generator
Location: C:\Program Files\MSSOAP\Binaries\MSSMOGen.dll
Language: Standard

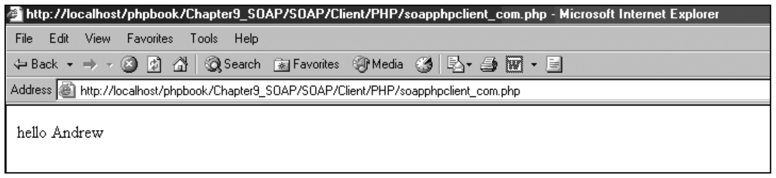
Figure 9.13 The Microsoft SOAP SDK COM library reference in Visual Basic.

As soon as the COM object is compiled, you can use it from within PHP:

```
<?php

//load SOAP client COM object
$soapob = new COM("php4winsoap.output");

//call getdata method to obtain result of SOAP exchange
$soapmessage = $soapob->getdata();

//output result
print($soapmessage);

?>
```

If you run this script, the web service output is displayed, as in the previous example (see Figure 9.14).

12 1690 CH09 4/25/02 2:01 PM Page 227


http://localhost/phpbook/Chapter9_SOAP/SOAP/Client/PHP/soapphpclient_com.php - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/phpbook/Chapter9_SOAP/SOAP/Client/PHP/soapphpclient_com.php

```
hello Andrew
```

Figure 9.14 Running your web service using a wrapped COM object.

Native PHP Implementation

PHP also lets you not use COM objects at all and connect and query WSDL files directly from PHP. This is made possible by the Simple Web Services API (SWSAPI). It is an open-source API whose creation has been led by the ActiveState Corporation in order to establish the same standard syntax for connecting and querying WSDL files in several different languages. SWSAPI currently is in beta and is available for Perl, Python, and PHP. It is expected to be made available for Ruby.

Using a native implementation means that you use your PHP libraries. In this case, the SWSAPI is a PHP library that builds on a native PHP implementation for SOAP called SOAP4X. To make use of the SWSAPI, you must unzip the PHP files into a directory you can access. You then make use of the SWSAPI via the following code:

```
<?php

require_once('webservice.php');

$soapob = WebService::ServiceProxy('http://localhost/phpbook/Chapter9
➥_SOAP/SOAP/Server/PHP4WINSOAP.WSDL');

$soapmessage = $soapob->HelloFunc("Andrew");

print($soapmessage);

?>
```

Here you load up the SWSAPI functions from the SWSAPI PHP library:

```
require_once('webservice.php');
```

What remains is very much like what you have seen using the Microsoft SOAP SDK COM objects. First, you call the WSDL file and store it in a variable:

```
$soapob = WebService::ServiceProxy('http://localhost/phpbook/Chapter9
➥_SOAP/SOAP/Server/PHP4WINSOAP.WSDL');
```

12 1690 CH09 4/25/02 2:01 PM Page 228

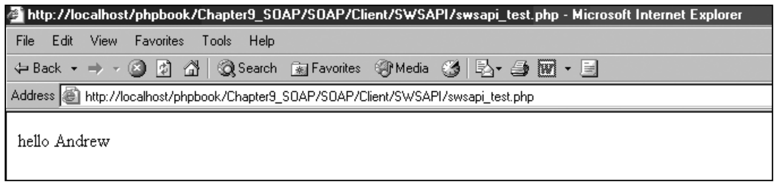
You then call a function of the web service and store the result in a variable:

```
$soapmessage = $soapob->HelloFunc("Andrew");
```

Finally, you display the result:

```
print($soapmessage);
```

If you run the script, you can see the result of calling your web service using the SWSAPI, as shown in Figure 9.15.


http://localhost/phpbook/Chapter9_SOAP/SOAP/Client/SWSAPI/swsapi_test.php - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/phpbook/Chapter9_SOAP/SOAP/Client/SWSAPI/swsapi_test.php

```
hello Andrew
```

Figure 9.15 Running your web service using the SWSAPI.

Further information, downloads, and the SWSAPI specification can be found at `http://aspn.activestate.com/ASPN/WebServices/SWSAPI/`.

Useful Tools

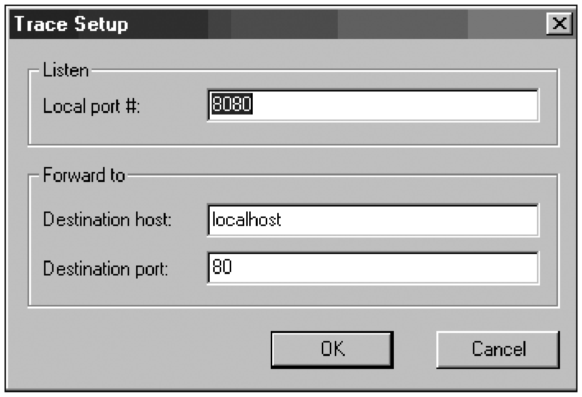
One of the most useful tools that the toolkit provides is the Trace utility (MsSoapT.exe). You access it by selecting Start, SOAP SDK. Trace lets you view SOAP message exchanges between client applications and the web service at either the web service or the client application side.

If you monitor the web service side, you must modify the service name portion of the WSDL file as follows:

```
<soap:address location='http://localhost:8080/phpbook/Chapter9
➥_SOAP/SOAP/Server/PHP4WINSOAP.ASP' />
```

Here you add a port number (8080) to the web service gateway's URL. If you start the Trace utility and select formatted trace, you are asked for the local port to listen on, as shown in Figure 9.16. In this case, because you are using port 8080, specify port 8080.

12 1690 CH09 4/25/02 2:01 PM Page 229


Trace Setup

Listen

Local port #: 8080

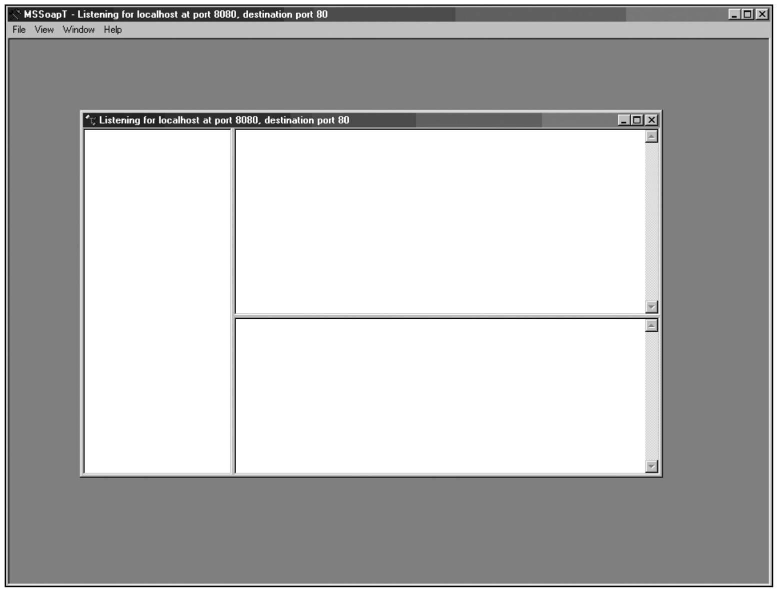
Forward to

Destination host: localhost

Destination port: 80

Figure 9.16 The Trace Setup dialog box.

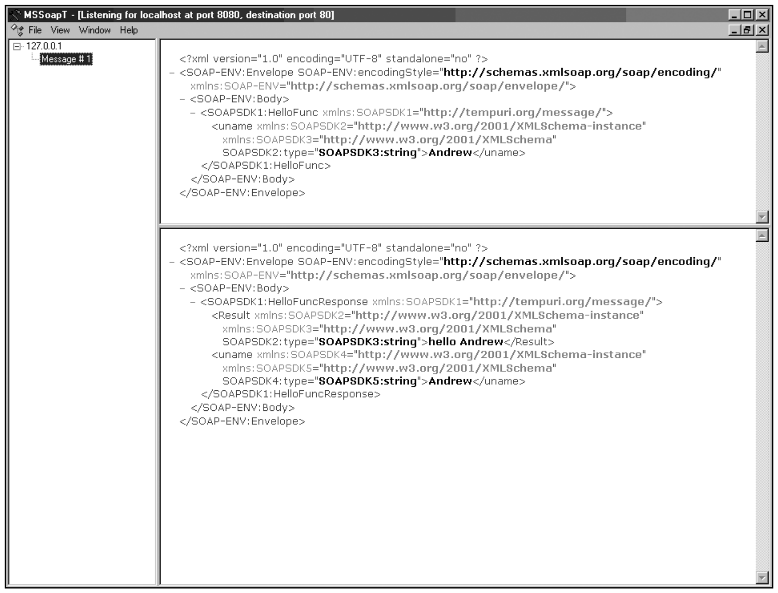
If you click OK to start the trace, you see the Trace window, shown in Figure 9.17.


MSSoapT - Listening for localhost at port 8080, destination port 80

Listening for localhost at port 8080, destination port 80

Figure 9.17 The SOAP Trace window.

If you run the web service client PHP script again, the Trace window stores the result of the SOAP message exchange. This exchange is stored under the IP that the web service was delivered from (in this case, the local host address of 127.0.0.1). The top pane of the Trace window contains the SOAP message that calls the web service's public function or method. The bottom pane contains the resulting SOAP message that the public function or method returns (see Figure 9.18). Note that if you change the WSDL file to support listening with the Trace utility and the Trace utility is not running, your web service reports an error.


MSSoapT - [Listening for localhost at port 8080, destination port 80]
File View Window Help
127.0.0.1
Message # 1

```
  <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <SOAPSDK1:HelloFunc xmlns:SOAPSDK1="http://tempuri.org/message/">
        <uname xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:SOAPSDK3="http://www.w3.org/2001/XMLSchema"
          SOAPSDK2:type="SOAPSDK3:string">Andrew</uname>
      </SOAPSDK1:HelloFunc>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

```
  <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <SOAPSDK1:HelloFuncResponse xmlns:SOAPSDK1="http://tempuri.org/message/">
        <Result xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:SOAPSDK3="http://www.w3.org/2001/XMLSchema"
          SOAPSDK2:type="SOAPSDK3:string">hello Andrew</Result>
        <uname xmlns:SOAPSDK4="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:SOAPSDK5="http://www.w3.org/2001/XMLSchema"
          SOAPSDK4:type="SOAPSDK5:string">Andrew</uname>
      </SOAPSDK1:HelloFuncResponse>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Figure 9.18 The SOAP Trace window displaying SOAP messages.

The Trace utility is very useful in helping you see what SOAP messages are exchanged between the web service and client applications. It therefore helps you debug any problems in your web services.

Summary

This chapter looked at what web services are, what comprises them (UDDI, WSDL, and SOAP), and how you can use PHP to create, look up, and query web services.