Chapter 3

Creating a Web Service

The last chapter demonstrated how to use SOAP to create a Web-service client. This chapter will focus on how a user can create a Web service. The environment utilized will be similar to the one covered for the SOAP client examples (see Amazon Web Services, page 19), including the Windows Server 2003, Perl, the SOAP::Lite Perl module, and IIS 6.0 running on an Intel-based server.

The example that will be demonstrated is fictional. Although I may, at some time, offer a Web service similar to the one discussed in this chapter, it is not currently a public Web service.

A User-Created Web Service for Searching/Retrieving Results from a Database of Libraries

This Web-service example will provide the ability to search a database of libraries and retrieve the results. The service is based on the database of libraries available in lib-web-cats (www.librarytechnology.org/libwebcats); this database includes information on more than twenty-five thousand libraries worldwide.

The Web service discussed in this chapter is designed to operate with two types of messages, the first to pass the service request to the server, and the second to pass the response back to the client. This is considered a *synchronous service*.

First, a plan for how the messages should be constructed needs to be established. The basic input for the Web service will invoke a method named *LibrarySearch* within the designated namespace *LibraryInformation Server*, and the input will pass the text to be searched as a string. I'll also include an optional parameter, which specifies the fields to search. The service request might be conceptualized in the simple XML fragment shown in figure 5.

The application will take the basic structure shown in figure 5 and transform it into a SOAP message, including all the headers needed; provide references to each of the schemas and namespaces involved; and specify the data types that apply. The resulting SOAP message would look something like the script shown in figure 6.

Compared to the service request, the service response will be more complex. Instead of passing a single value and qualifier, the response will carry much more information. After the service provider carries out the LibrarySearch method, a user may want to receive some information about each of the items returned in the search; the response will also tell the user the number of items returned.

Such a response requires the user to define a complex data structure. One component of the results will consist of an object that tells him or her about each library found by the query. The structure will be called *item* and will include several descriptive elements. The item structure can be conceptualized as the script shown in figure 7.

The user, however, will receive many of these, so he or she will need to build an array of <item>. This array is deemed *ResultElements*, and it will hold as many instances of <ResultElement> as the user needs. The result set will also tell the user the number of items to be returned. This value will be passed in a data structure called *TotalResultsCount*. So, the request response will consist of a complex data structure called <return>, which includes <ResultElements> and <TotalResultsCount> where ResultElements is an array of multiple <ResultElement>. As the application wraps the results in SOAP, it will look something like the script shown in appendix 13.

I have described the design of the service, so now I'll describe how to construct a WSDL file, which will render the data structures involved and the methods and messages available. This WSDL file will include each of the concepts covered in chapter one.

Because the WSDL file is an XML document (see appendix 14), it begins with the appropriate declaration. The root element of the document will be <*definitions*> as are all WSDL files. The <*types*> section specifies the data structures involved in the service, and you can see the structure outline for the complex object (described above). The WSDL includes a section that defines the <messages>, which include one for the service request called <*LibrarySearch*> and one for the service response called <*LibrarySearchResponse*>. The <*PortType*> section references the previously defined messages. The <*bindings*> and <*service*> provide the specifications for how this service can be invoked. Note that this listing is meant to illustrate the structure of WSDL and should not be considered the specification of a publicly available service.

Now that the user has determined what he or she wants the service to do, the user can write some programs to implement it. I'll continue in the mode of using Perl with SOAP::Lite.

The most complex part of the programming project will be to create a Perl module that implements the service. This program will need to access the underlying database and be able to deliver results to the SOAP environment. The Perl module illustrated in appendix 15 performs these tasks.

This module uses the Open Database Connectivity (ODBC) model to communicate with the underlying database. The ODBC layer is provided by the Win32::ODBC

```
<LibrarySearch>
  <query>Vanderbilt University</query>
  <SearchType>LibraryName</SearchType>
</LibrarySearch>
```

Figure 5:
Simple XML Fragment—Service Request in LibrarySearch Method

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:LibrarySearch xmlns:ns1="urn:LibraryInformationServer"
     SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <query xsi:type="xsd:string">Vanderbilt University</query>
      <SearchType xsi:type="xsd:string">LibraryName</SearchType>
    </ns1:LibrarySearch t>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 6:
SOAP Message Generated from Simple XML Fragment in Figure 5

```
<item>
  <Institution>Vanderbilt University</Institution>
  <Address>419 21st Avenue South</Address>
  <City>Nashville</City>
  <State>TN</State>
  <Zip>37240</Zip>
  <Country>United States</Country>
  <URL">http://www.library.vanderbilt.edu</URL>
  <OPAC>http://acorn.library.vanderbilt.edu</OPAC>
  <ILS>Unicorn</ILS>
  <LibraryType>Academic</LibraryType>
</item>
```

Figure 7:
Item Structure

Perl module and the ODBC driver of the underlying database. Using ODBC makes the program database independent, and it can be used with any database with an ODBC driver available.

The Perl module contains the sub-routine *LibrarySearch,* which serves as the primary method for the user-created Web service. The module receives as its input parameters the messages passed by the service request. The <*query*> parameter is passed into the sub-routine, into the local variable *$request*, which is then inserted into an SQL statement. The SQL statement is stored as a string in the global variable *$SqlStatement*, and it's executed by the ODBC engine by the sub-routine *executeSQL*. ODBC returns the results of the query in the object *$db*. The user can then use the ODBC method *FetchRow* to retrieve each of the table rows returned by the query. An associative array named *data* holds the values for each field. As each row is processed, the values in the data array are pushed into the SOAP data structures. The rows are counted as it goes along, so it can also return the number of results in the *TotalResultsCount* component of the structured response.

The user must ensure that all data passed in the service response is valid XML. The *MakeValidXML* sub-routine performs this function. In the simple example, it only fixes the known problem of unescaped ampersands. In a production service, the user would need to perform a rigorous check of the data to ensure the service does not fail due to problems with the data (such as using characters outside the defined encoding or character set).

Once the module has been developed that performs the work defined in the service, the user can activate it through a program that sets up the server to monitor SOAP requests. There are many ways to activate a SOAP service; in the SOAP::Lite environment, for example, a user can establish a service with the few simple lines of code (appendix 16), and the service operates through TCP sockets.

With the service operational, the user can use the following command-line client (appendix 17) to access it, which continues to rely on Perl and SOAP::Lite.

Although it is clearly more complex to develop a Web service than to access an existing one, it can be accomplished without too much effort (as illustrated in this chapter). This Web service performs a relatively simple task—that of searching a database and delivering a result. Most of the complexity comes into play in the setting up the data types and message structures needed for the SOAP environment, and many programming environments automate much of this part of the development process.

Appendix 13:
<ResultElement> Array

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
  <ns1:LibrarySearchResult
   xmlns:ns1="urn:LibrarySearch"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <return xsi:type="ns1:LibraryInformationServer">
     <resultElements xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/"
      xsi:type="ns3:Array" ns3:arrayType="ns1:ResultElement[3]">
       <item xsi:type="ns1:ResultElement">
         <LibraryName xsi:type="xsd:string">Vanderbilt University
          Library</LibraryName>
         <Institution xsi:type="xsd:string">Vanderbilt University</Institution>
         <Address xsi:type="xsd:string">419 21st Avenue South</Address>
         <City xsi:type="xsd:string">Nashville</City>
         <State xsi:type="xsd:string">TN</State>
         <Zip xsi:type="xsd:string">37240</Zip>
         <Country xsi:type="xsd:string">United States</Country>
         <URL xsi:type="xsd:string">http://www.library.vanderbilt.edu</URL>
         <OPAC xsi:type="xsd:string">http://acorn.library.vanderbilt.edu</OPAC>
         <ILS xsi:type="xsd:string">Unicorn</ILS>
      <LibraryType xsi:type="xsd:string">Academic</LibraryType>
    </item>

    <item xsi:type="ns1:ResultElement">
      <LibraryName xsi:type="xsd:string">Sarah Shannon Stevenson Science and
               Engineering Library</LibraryName>
      <Institution xsi:type="xsd:string">Vanderbilt University</Institution>
      <Address xsi:type="xsd:string">3200 Stevenson Center,
    Box 1803 Station B</Address>
      <City xsi:type="xsd:string">Nashville</City>
      <State xsi:type="xsd:string">TN</State>
      <Zip xsi:type="xsd:string">37240</Zip>
      <Country xsi:type="xsd:string">United States</Country>
      <URL xsi:type="xsd:string">http://www.library.vanderbilt.edu/science/</URL>
      <OPAC xsi:type="xsd:string">http://acorn.library.vanderbilt.edu</OPAC>
      <ILS xsi:type="xsd:string">Unicorn</ILS>
      <LibraryType xsi:type="xsd:string">Academic</LibraryType>
    </item>

    <item xsi:type="ns1:ResultElement">
      <LibraryName xsi:type="xsd:string">Walker Management
          Library</LibraryName>
      <Institution xsi:type="xsd:string">Vanderbilt University</Institution>
      <Address xsi:type="xsd:string">401 21st Avenue South</Address>
      <City xsi:type="xsd:string">Nashville</City>
      <State xsi:type="xsd:string">TN</State>
      <Zip xsi:type="xsd:string">37240</Zip>
      <Country xsi:type="xsd:string">United States</Country>
      <URL xsi:type="xsd:string">http://www.library.vanderbilt.edu/walker/</URL>
      <OPAC xsi:type="xsd:string">http://acorn.library.vanderbilt.edu</OPAC>
      <ILS xsi:type="xsd:string">Unicorn</ILS>
      <LibraryType xsi:type="xsd:string">Academic</LibraryType>
    </item>

   </resultElements>
   <TotalResultsCount xsi:type="xsd:int">3</TotalResultsCount>
  </return>
 </ns1:LibrarySearchResult>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Appendix 14:
WSDL File for User-Created Web Service

```
<?xml version=”1.0” encoding=”UTF-8”?>

<!– WSDL description of the LibrarySearch API. –>

<!– Revision 2006-02-01 by Marshall Breeding–>


<definitions name=”LibrarySearch”
       targetNamespace=”urn:LibrarySearch”
       xmlns:typens=”urn:LibrarySearch”
       xmlns:xsd=”http://www.w3.org/2001/XMLSchema”
       xmlns:soap=”http://schemas.xmlsoap.org/wsdl/soap/”
       xmlns:soapenc=”http://schemas.xmlsoap.org/soap/encoding/”
       xmlns:wsdl=”http://schemas.xmlsoap.org/wsdl/”
       xmlns=”http://schemas.xmlsoap.org/wsdl/”>

 <!– Types for LibrarySearch –>

 <types>
  <xsd:schema xmlns=”http://www.w3.org/2001/XMLSchema”
        targetNamespace=”urn:LibrarySearch”>
   <xsd:complexType name=”LibrarySearchResponse”>
    <xsd:all>
     <xsd:element name=”resultElements”       type=”typens:ResultElementArray”/>
     <xsd:element name=”TotalResultsCount”      type=”xsd:int”/>
    </xsd:all>
   </xsd:complexType>

   <xsd:complexType name=”ResultElement”>
     <xsd:all>
       <xsd:element name=”LibraryName”       type=”xsd:string” />
       <xsd:element name=”Institution”       type=”xsd:string” />
       <xsd:element name=”Address”         type=”xsd:string” />
       <xsd:element name=”City”          type=”xsd:string” />
       <xsd:element name=”State”          type=”xsd:string” />
       <xsd:element name=”Zip”           type=”xsd:string” />
       <xsd:element name=”Country”         type=”xsd:string” />
       <xsd:element name=”URL”           type=”xsd:string” />
       <xsd:element name=”OPAC”          type=”xsd:string” />
       <xsd:element name=”ILS”           type=”xsd:string” />
       <xsd:element name=”LibraryType”       type=”xsd:string” />
     </xsd:all>
   </xsd:complexType>
   <xsd:complexType name=”ResultElementArray”>
    <xsd:complexContent>
     <xsd:restriction base=”soapenc:Array”>
       <xsd:attribute ref=”soapenc:arrayType”
         wsdl:arrayType=”typens:ResultElement[]”/>
     </xsd:restriction>
    </xsd:complexContent>
   </xsd:complexType>
  </xsd:schema>
 </types>
```

```
<!– Messages for LibrarySearch API –>

<message name=”LibrarySearch”>
 <part name=”query”                  type=”xsd:string”/>
 <part name=”SearchType”             type=”xsd:string”/>
</message>
<message name=”LibrarySearchResult”>
 <part name=”return”                 type=”typens:LibrarySearchResponse”/>
</message>

<!– Port for APIs “LibrarySearch” –>

<portType name=”LibrarySearchPort”>
 <operation name=”LibrarySearch”>
  <input message=”typens:LibrarySearchRequest”/>
  <output message=”typens:LibrarySearchResponse”/>
 </operation>
</portType>

<!– Binding for LibrarySearch API - RPC, SOAP over HTTP –>

<binding name=”LibrarySearchBinding” type=”typens:LibrarySearchPort”>
 <soap:binding style=”rpc”
       transport=”http://schemas.xmlsoap.org/soap/http”/>
 <operation name=”LibrarySearch”>
  <soap:operation soapAction=”urn:LibrarySearch”/>
  <input>
   <soap:body use=”encoded”
        namespace=”urn:LibrarySearch”
        encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”/>
  </input>
  <output>
   <soap:body use=”encoded”
        namespace=”urn:LibrarySearch”
        encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”/>
  </output>
 </operation>
</binding>

<!– Endpoint for LibrarySearch –>

<service name=”LibrarySearchService”>
 <port name=”LibrarySearchPort” binding=”typens:LibrarySearchBinding”>
  <soap:address location=”http://www.librarytechnology.org:8001”/>
 </port>
</service>
</definitions>
```

Appendix 15:
Perl Module That Implements the User-Created Web-Service

```
package LibraryInformationServer;
use Win32::ODBC;
Win32::ODBC::ODBCLevel( SQL_OV_ODBC3 );
$createODBC;

my $DSN = “BREEDING”;
my $NS = “http://www.librarytechnology.org/LibrarySearch”;

my @KeyFields = (
 “LibraryName”,
 “Institution”,
 “Address”,
 “City”,
 “State”,
 “ZipCode”,
 “Country”,
 “URL”,
 “OPAC”,
 “ILS”,
 “Type”
);

sub LibrarySearch {
   $db = new Win32::ODBC($DSN);
   local($method,$request,$type) = @_;
   local $LibraryWebSite = {};
   local @LibraryElements = {};
   my $SqlStatement = “SELECT * FROM libwebcats
                  WHERE (LibraryName CT \’$request\’)
                       OR (Institution CT \’$request\’)
                  ORDER BY LibraryName”;
   &executeSQL($SqlStatement);
   print “SQL: $SqlStatement\n”;
   if ($type eq “Website”) {
   } elsif ($type eq “Website”) {
     $ResultType = “LibraryWeb”;
   } elsif ($type eq “OPAC”) {
     $ResultType = “OnlineCatalog”;
   } else {
     $ResultType = “LibraryWeb”;
   }
   print “Type: $type\n”;
   my $i = 0;
   while ($db->FetchRow()) {
     my(%data) = $db->DataHash();
     print “Library: $data{‘LibraryName’}\n”;
     foreach $fld (@KeyFields) {
        $data{$fld} = &MakeValidXML($data{$fld});
     }

     $ItemList[$i] = SOAP::Data->name(‘item’ => \SOAP::Data->value(
         SOAP::Data->type(‘xsd:string’)->name(‘LibraryName’)
```

```
                ->value($data{'LibraryName'}),
            SOAP::Data->type('xsd:string')->name('Institution')
                 ->value($data{'Institution'}),
            SOAP::Data->type('xsd:string')->name('Address')->value($data{'Address'}),
            SOAP::Data->type('xsd:string')->name('City')->value($data{'City'}),
            SOAP::Data->type('xsd:string')->name('State')->value($data{'State'}),
            SOAP::Data->type('xsd:string')->name('Zip')->value($data{'ZipCode'}),
            SOAP::Data->type('xsd:string')->name('Country')->value($data{'Country'}),
            SOAP::Data->type('xsd:string')->name('URL')->value($data{'LibraryWeb'}),
            SOAP::Data->type('xsd:string')->name('OPAC')->value($data{'OnlineCatalog'}),
            SOAP::Data->type('xsd:string')->name('ILS')->value($data{'ILS'}),
            SOAP::Data->type('xsd:string')->name('LibraryType')->value($data{'Type'})
      ))
      ->uri("$NS")
      ->prefix("ns1")
      ->type("ns1:ResultElement");
      $i++;
      last if ($i > 5);
   }
   $db->Close();
#   return "The $type is $URL";
   my $resultElements = SOAP::Data->name('resultElements' => \SOAP::Data
            ->value(@ItemList))
                  ->uri("$NS")
                  ->type("ns3:ResultElementArray")
                  ->prefix("ns3");
   my $resultCount = SOAP::Data->type('xsd:int')->name('TotalResultsCount')->value($i);
   return $resultElements,$resultCount;
}

sub executeSQL {
  ($SQLStatement) = @_;
  if ($db->Sql($SQLStatement)) {
      open (ERRORLOG,">>$logdirectory\\$errorlog");
      ($ErrNum, $ErrText, $ErrConn) = $db->Error();
      $ErrorData = "$ErrNum, $ErrText, $ErrConn";
      print "ERROR: $ErrorData\n";
      $db->Close();
   }
}


sub MakeValidXML {
  local ($value) = @_;
  $value =~ s/\&/&amp;/g;
  return $value;
}
```

Appendix 16:
Code for User-Created Web Service in a SOAP::Lite Environment

```
use SOAP::Lite +trace;
use SOAP::Transport::TCP;

my $daemon = SOAP::Transport::TCP::Server
  ->new(LocalAddr => 'localhost', LocalPort => 8001, Listen => 5);
$daemon->dispatch_to('LibraryInformationServer');
print "SOAP TCP server listening...\n";
print "  Host: ", $daemon->sockhost, "\n";
print "  Port: ", $daemon->sockport, "\n";
$daemon->handle();
```

Appendix 17:
Command Line Client to Access User-Created Web Service

```
use SOAP::Lite;

my $q="Marshall Breeding";
my $LibrarySearch = SOAP::Lite -> service("http://localhost/LibrarySearch.wsdl");

my $results = $LibrarySearch -> LibrarySearch($q,"LibraryName");
print "About $results->{'TotalResultsCount'} results.\n";

foreach my $result (@{$results->{resultElements}}) {
  print "\n",
  print $result->{URL}."'>".$result->{LibraryName}."</a>\n";
  print $result->{Address},"\n";
  print"\n";
}
```