Describing a Web Service with the Web Services Description Language (WSDL)

NEW TERM One of the big advantages that Web services have over the more traditional approaches is that they can be described formally in a document called a *Web Service Description Language (WSDL) document* or just WSDL. A WSDL document is an XML document that provides all the information that you need to connect to the Web service. In addition, it contains some of the data that you need to evaluate whether this Web service can fulfill your requirements. The Universal Description, Discovery, and Integration (UDDI) document provides the rest of this information. We will cover the UDDI in Hour 11, "Advertising a Web Service."

In this hour, you will learn how to create the WSDL for a Web service. You will learn how to format it so that a client program can use it to generate the code that it needs to connect to this service.

In this hour, you will learn

- What WSDL is
- Why WSDL is needed
- What information is stored in the WSDL document
- How WSDL is used in a Web service transaction

The WSDL Document

The WSDL document is formatted to the XML specification. In addition, it is authored with a specific XML grammar that was devised to communicate metadata about a Web service in a uniform manner to all potential clients. In other words, a WSDL document is an XML document that conforms to a specification. All the metadata about the Web service is contained somewhere in this file. The structure is there to make it easy to figure out what the data means.

XML is a good choice for this type of application because it is human readable, but it is also precise enough to be machine readable. All a human has to do is learn the meaning of the WSDL element tags, and she can read the document just as you are reading this paragraph. All a programmer has to do is to use an XML parser to extract the data into local variables. The program will have all the details it needs to generate the code necessary to connect to the Web service.

We mentioned earlier that the WSDL contains some of the information needed to select a Web service that fits your requirements. It follows, then, that it also contains some of the information needed to publish your Web service's capabilities to a registry. Figure 10.1 shows how a WSDL is used to find, bind, and publish the Web service.

Any program that wants to use a Web service uses the WSDL to figure out how to bind to it. The Web service author creates the WSDL—either by hand or using a tool that generates it. He publishes it by sending it (or its URL) to a directory service, which can also be called a registry. Web service shoppers use the registry to identify a Web service that meets their needs.

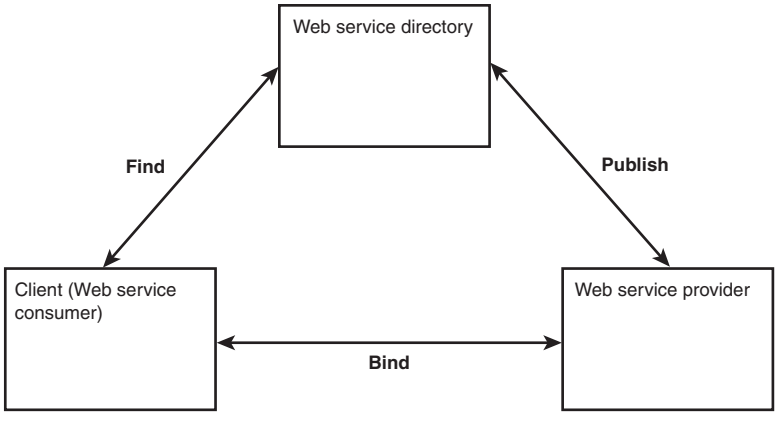

FIGURE 10.1
The basic operations that use the WSDL are find, bind, and publish.

The Concrete and Abstract Description

The WSDL document is subdivided logically into two different groupings—the concrete and the abstract descriptions. These can also be called the functional and the nonfunctional descriptions. The concrete description is composed of those elements that are oriented toward binding the client to the service physically. The concrete description performs the same tasks as the Interface Definition Language (IDL) in CORBA. The abstract description is composed of those elements that are oriented toward describing the capabilities of the Web service.

The four abstract XML elements that can be defined in a WSDL are as follows:

- `<wsdl:types>`
- `<wsdl:message>`
- `<wsdl:operation>`
- `<wsdl:portType>`

In addition, there are three concrete XML elements in a WSDL:

- `<wsdl:service>`
- `<wsdl:port>`
- `<wsdl:binding>`

In addition to these, you will also see SOAP messages and XML schema definitions in the WSDL. Hour 7, "Understanding XML," discusses the special elements used for describing XML schemas. The SOAP-specific elements are covered in Hour 9, "Exchanging Messages with SOAP."

The `types` Element

New Term The *<wsdl:types>* element is used to indicate that a WSDL type is being declared. One of the rules of the SOAP specification is that only one input and one output is allowed in the messages that are sent across the Internet from one computer to another. (This one value can be a complex type such as an array, however.) The reason for this is to maintain the simplicity of the communications by making the two computers that are communicating (the endpoints) do the work of parsing the parameters.

You can overcome this limitation by designing the messages that you send to always send one primitive value (that is, `string` or `int`) period, which is a very limiting solution. The best way to overcome this limitation is to create a user-defined data type by using the `<wsdl:types>` element. A user-defined data type is a variable that is composed of primitive data types. These types are the equivalent of C++ structs or Java classes that contain only data and no methods.

The following snippet shows us a customer definition that we can use to illustrate how this works:

```
<customer>
   <customerID>1001</customerID>
   <lastName>Maddox</lastName>
   <firstName>Greg</firstName>
   <address> 123 First Street</address>
   <city>Atlanta</city>
   <state>GA</state>
   <zip>30003</zip>
</customer>
```

This data is kept very simple in order to keep our focus on the process of creating the WSDL, not on data-modeling issues. We can create an XML schema definition of this data, which we will use later when creating the WSDL. Listing 10.1 shows this schema.

Listing 10.1 The Customer Schema

```
<xsd:schema targetNamespace="http://www.stevepotts.com/customer.xsd"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.stevepotts.com/customer.xsd">
  <xsd:element name="customer">
     <xsd:complexType >
        <xsd:sequence>
           <xsd:element name="customerID" type="xsd:string"/>
           <xsd:element name="lastName" type="xsd:string"/>
           <xsd:element name="firstName" type="xsd:string"/>
           <xsd:element name="address" type="xsd:string"/>
           <xsd:element name="city" type="xsd:string"/>
```

LISTING 10.1 continued

```
            <xsd:element name="state" type="xsd:string"/>
            <xsd:element name="zip" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

The first line specifies the string that an XML document must use to specify that it must conform to this schema:

```
<xsd:schema targetNamespace="http://www.stevepotts.com/customer.xsd"
```

The namespace for the schema, `xsd`, is declared next:

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

The root element is called `"customer"`:

```
<xsd:element name="customer">
```

A `complexType` is a user-defined type. It is similar to a C++ struct.

```
<xsd:complexType name="customerType">
```

The element sequence indicates that all the elements defined next are required to be provided in exactly this sequence. Alternatively, we could have used the `<xsd:all>` tag if order were not important:

```
<xsd:sequence>
```

Each one of the fields that we want to include gets its own element definition. The name will be the name of the data field, and the type will be the simple data type of that field. All of these are of type `xsd:string`:

```
<xsd:element name="customerID" type="xsd:string"/>
<xsd:element name="lastName" type="xsd:string"/>
<xsd:element name="firstName" 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"/>
```

Now that we have a schema, we can create the `<wsdl:types>` section of the WSDL document that we are building. Listing 10.2 shows this section.

LISTING 10.2 The `wsdl:types` Section

```
<wsdl:types>
   <xsd:schema targetNamespace="http://www.stevepotts.com/customerType.xsd"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <xsd:element name="customer">
        <xsd:complexType>
           <xsd:sequence>
              <xsd:element name="customerID" type="xsd:string"/>
              <xsd:element name="lastName" type="xsd:string"/>
              <xsd:element name="firstName" 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:sequence>
        </xsd:complexType>
     </xsd:element>
   </xsd:schema>
</wsdl:types>
```

If you want to include tags from multiple schemas, you can refer to those tags by adding additional namespaces to the `<xsd:schema>` element. Hour 7 provides a detailed discussion of namespaces.

As you can see, the `wsdl:types` element is created directly by adding the `<wsdl:types>` and `</wsdl:types>` tags to the schema definition for this user-defined data type.

The reason for doing this is simple. The XML schema provides exactly the data needed to communicate the format of a data type.

The `message` Element

NEW TERM The next important element in the WSDL is the *`message`*. Messages are one-way communications from one computer to another. For example, in the typical request/response scenario, one message is sent and a second message is received.

The `message` element is considered an abstract element because it describes the message logically instead of physically. By reading the message, a client or potential client can get a thumbnail sketch of what this Web service can provide in the way of processing.

If we decided to create a message called `addCustomer`, we would add a `message` element to the WSDL called `addCustomer`, as shown here:

```
<wsdl:message name="addCustomer">
   <wsdl:part name="customerInfo" element="tns:customer"/>
<wsdl:/message>
```

This message is going to add a customer to the Web service by sending it an instance of the `customer` element that we defined in the `types` element. At the detail level, the message is really going to send customer information to the Web service, and the Web service will perform the calls to the back-end system. The Web service is just an interface to the back-end system.

```
 <wsdl:message name="confirmation">
   <wsdl:part name="response" element="xsd:integer"/>
<wsdl:/message>
```

This message will send an integer back that confirms that the `addCustomer` message successfully completed.

```
<wsdl:message name="exceptionMessage">
   <wsdl:part name="badResult" element="xsd:integer"/>
<wsdl:/message>
```

This message will send an integer back that tells the client that the `addCustomer` message did not successfully complete.

The `operation` Element

NEW TERM The *`operation`* element is analogous to a method call in Java or a subroutine call in Visual Basic. One difference is that only three messages are allowed in an operation:

- **The Input Message**—Defines the data that the Web service expects to receive.
- **The Output Message**—Defines the data that the Web service expects to send in response.
- **The Fault Message**—Defines the error messages that can be returned by the Web service.

Several types of operations can be declared in a WSDL document. They are

- **Request/Response**—A client makes a request, and the Web service responds to it.
- **Solicit/Response**—A Web service sends a message to the client, and the client responds.
- **One-way**—A client sends a message to the Web service but expects no response.
- **Notification**—A Web service sends a message to the client but expects no response.

Figure 10.2 shows these four operation types graphically.

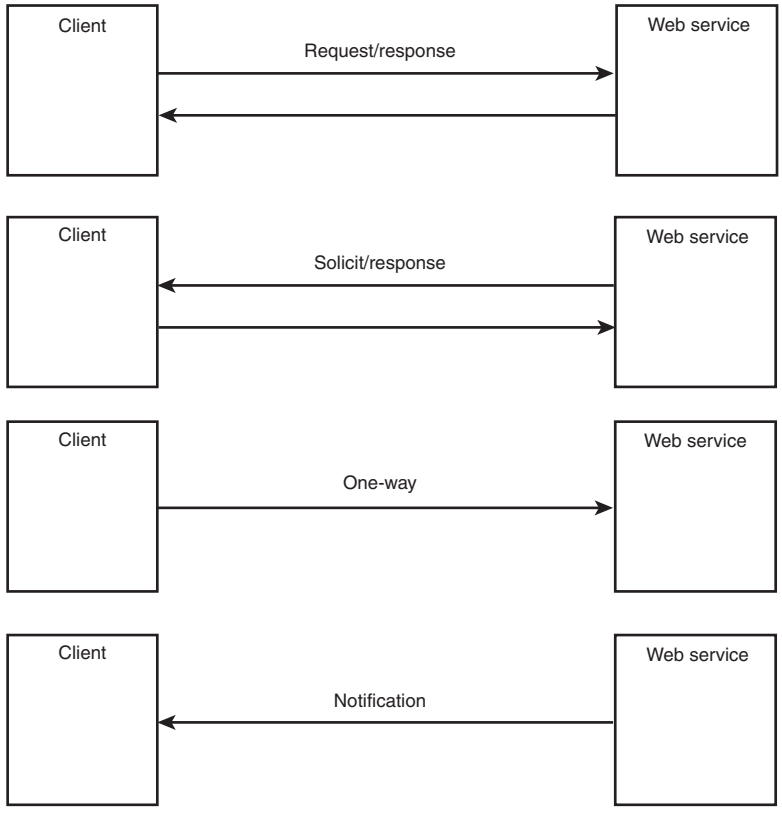

FIGURE 10.2
The basic operations are request/response, solicit/response, one-way, and notification.

The syntax of an operation is simple. If the operation is a request/response type, the format will be as shown here:

```
<wsdl:operation name="createNewCustomer">
   <wsdl:input message="addCustomer">
   <wsdl:output message="confirmation">
   <wsdl:fault message="exceptionMessage">
<wsdl:/operation>
```

If the message is of the solicit/response type, the output element will appear before the input element. If it is a one-way message, there is no output. If it is of the notification type, there will be no input.

The `portType` Element

NEW TERM The *`portType`* is one of the oddly named elements in the WSDL. A port, in Web services jargon, is a single Web service. The `portType`, then, is the set of all operations that one Web service can accept. The `portType` is an abstract element that

doesn't provide information on how to connect directly to a Web service. It does provide a one-stop point where a client can obtain information on all the operations that a Web service provides.

The syntax for a `portType` is very simple:

```
<wsdl:portType name="newCustomerPortType">
   <wsdl:operation name="createNewCustomer">
      <wsdl:input message="addCustomer"/>
      <wsdl:output message="confirmation"/>
      <wsdl:fault message="exceptionMessage"/>
   </wsdl:operation>
</wsdl:portType>
```

As you can see, it is simply a container element for operation elements.

The `binding` Element

The elements that we have covered thus far have all been involved with specifying what a Web service is capable of providing in the way of functionality. From this point on, we will be focusing on how to obtain the information needed to physically connect to the Web service.

The *`binding`* element serves two purposes. First, it serves as the link between the abstract elements and the concrete elements in the WSDL. One of the attributes of the `binding` tag is the name of the `portType`. The second purpose that the `binding` element serves is to provide a container for information such as the protocol and the address of the Web service. The syntax for a `binding` element is shown in Listing 10.3.

NEW TERM

LISTING 10.3 The `newCustomerBinding` Section

```
<wsdl:binding name="newCustomerBinding" type="newCustomerPortType">
   <soap:binding style="rpc"
             transport="http://schemas.xmlsoap.org/soap/http" />
   <wsdl:operation name="createNewCustomer">
      <soap:operation
            soapAction="http://www.stevepotts.com/createNewCustomer"/>
         <wsdl:input>
           <soap:body use="encoded"
               namespace="http://www.stevepotts.com/customer"
               encodingStyle=
                    "http://schemas.xmlsoap.org/soap/encoding/"/>
         <wsdl:input>
         <wsdl:output>
           <soap:body use="encoded"
               namespace="http://www.stevepotts.com/customer"
               encodingStyle=
```

LISTING 10.3 continued

```
                    "http://schemas.xmlsoap.org/soap/encoding/"/>
         </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
```

The first line connects the `binding` with the `portType` that we created earlier in this hour. The same `portType` can appear in more than one `binding` element. For example, you could have a `binding` for SMTP as well. A client would only use one of the `binding`s when communicating with the Web service.

```
<wsdl:binding name="purchaseBinding" type="newCustomerPortType">
```

This `binding` is designated to be a SOAP `binding`. In addition, it is going to use the HTTP to send the SOAP documents.

```
   <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
```

The `soap:body` element provides details about how the operation is to be encoded. This example states that it will use encoding and specifies the URI of the `encodingStyle`.

```
          <soap:body use="encoded"
              namespace="http://www.stevepotts.com/customer"
              encodingStyle=
                   "http://schemas.xmlsoap.org/soap/encoding/"/>
```

Notice that both the input and the output use the same encoding scheme in this example. SOAP encoding is covered in Hour 9.

The `port` Element

NEW TERM The only piece of information that is now missing is the actual IP address and port of the Web service that is represented by this WSDL. The *`port`* element is where this information is located. The syntax of the `port` element is shown here:

```
<wsdl:port binding="newCustomerBinding" name="newCustomerPort">
   <soap:address
        location="http://www.stevepotts.com:1776/soap/servlet/rpcrouter">
</wsdl:port>
```

The address shown here is fictitious, but in a running example, this URL must be a real Web service handler.

The `service` Element

NEW TERM The most bizarrely named element of all is *`service`*. Many Web services are gathered together and called a service! This element is a container for all ports that are represented by a WSDL document. The ports within a service can't be chained so that the output of one port is the input to another. As a result, the `service` tag is of limited value, but is required by the specification.

```
<wsdl:service name="newCustomerService">
   <wsdl:documentation>
      This is for adding new customers.
   </wsdl:documentation>
   <wsdl:port binding="newCustomerBinding" name="newCustomerPort">
   <soap:address
        location="http://www.stevepotts.com:1776/soap/servlet/rpcrouter"/>
   </wsdl:port>
</wsdl:service>
```

Through the `port` element, the `service` has access to all the information available in the rest of the document.

The `definitions` Element

NEW TERM The root element in a WSDL document is *`wsdl:definitions`*. It contains elements to specify the `targetNameSpace`, as well as a number of ordinary namespaces to help keep out naming conflicts. The syntax of the `definitions` element is shown here:

```
<wsdl:definitions name="customerExample"
targetNamespace="http://www.stevepotts.com/customer.wsdl"
   xmlns:soap="http://www.schemas.xmlsoap.org/wsdl/soap/"
   xmlns:wsdl="http://www.schemas.xmlsoap.org/wsdl/"
   xmlns="http://www.stevepotts.com/customer.xsd">
```

The rest of the WSDL document appears under this element. The end of the document is indicated by

```
</wsdl:definitions>
```

The namespaces that are defined in this element are global to the WSDL document. Other namespaces can appear locally, so be aware of them. Figure 10.3 shows all these elements and their relationships.

The information in the WSDL document is adequate not only for the human reader, but also for a program to generate code to connect physically to the Web service. This automatic code generation is one of the outstanding features of the Web services architecture.

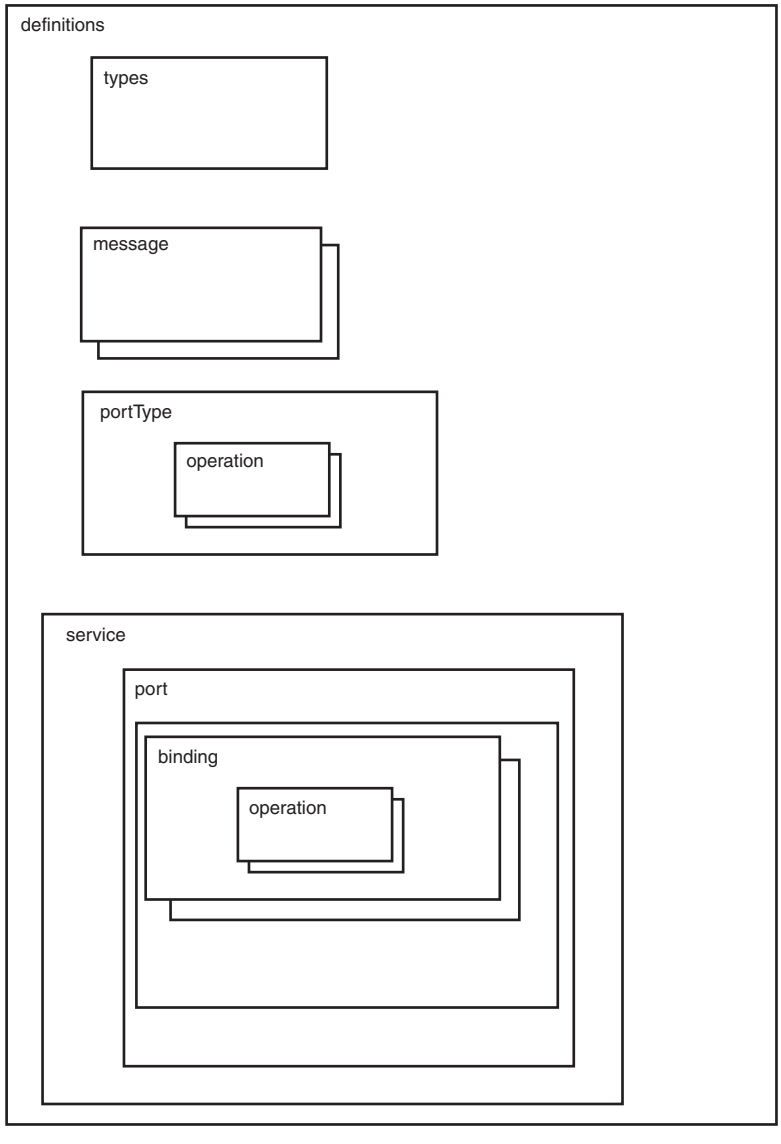

FIGURE 10.3
The elements of the WSDL document connect together to provide a complete picture of the Web service.

Listing 10.4 shows the entire customer WSDL document.

LISTING 10.4 The `customerExample` WSDL Document

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="customerExample"
targetNamespace="http://www.stevepotts.com/customer.wsdl"
```

LISTING 10.4 continued

```
    xmlns:soap="http://www.schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://www.schemas.xmlsoap.org/wsdl/"
    xmlns="http://www.stevepotts.com/customer.xsd">

<wsdl:types>
   <xsd:schema targetNamespace="http://www.stevepotts.com/customerType.xsd"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <xsd:element name="customer">
        <xsd:complexType>
           <xsd:sequence>
              <xsd:element name="customerID" type="xsd:string"/>
              <xsd:element name="lastName" type="xsd:string"/>
              <xsd:element name="firstName" 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:sequence>
        </xsd:complexType>
     </xsd:element>
   </xsd:schema>
</wsdl:types>
<wsdl:message name="addCustomer">
   <wsdl:part name="customerInfo" element=" customer"/>
</wsdl:message>
<wsdl:message name="confirmation">
   <wsdl:part name="response" element="xsd:integer"/>
</wsdl:message>
<wsdl:message name="exceptionMessage">
   <wsdl:part name="badResult" element="xsd:integer"/>
</wsdl:message>
<wsdl:portType name="newCustomerPortType">
   <wsdl:operation name="createNewCustomer">
      <wsdl:input message="addCustomer"/>
      <wsdl:output message="confirmation"/>
      <wsdl:fault message="exceptionMessage"/>
   </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="newCustomerBinding" type="newCustomerPortType">
   <soap:binding style="rpc"
             transport="http://schemas.xmlsoap.org/soap/http" />
   <wsdl:operation name="createNewCustomer">
      <soap:operation
            soapAction="http://www.stevepotts.com/createNewCustomer"/>
         <wsdl:input>
           <soap:body use="encoded"
               namespace="http://www.stevepotts.com/customer"
               encodingStyle=
```

LISTING 10.4 continued

```
                    "http://schemas.xmlsoap.org/soap/encoding/"/>
          </wsdl:input>
          <wsdl:output>
            <soap:body use="encoded"
                namespace="http://www.stevepotts.com/customer"
                encodingStyle=
                    "http://schemas.xmlsoap.org/soap/encoding/"/>
          </wsdl:output>
   </wsdl:operation>
</wsdl:binding>

<wsdl:service name="newCustomerService">
   <wsdl:documentation>
      This is for adding new customers.
   </wsdl:documentation>
   <wsdl:port binding="newCustomerBinding" name="newCustomerPort">
   <soap:address
        location="http://www.stevepotts.com:1776/soap/servlet/rpcrouter"/>
   </wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

When viewed in its entirety, the WSDL document represents a full-blown description of the Web service—complete with all the information needed to bind your client program to it. This information is so complete that many vendors have written software that can generate client programs from it.

Summary

This hour has shown you how Web services describe themselves. These descriptions are encoded in a special XML file called the WSDL document. You learned what the role of the WSDL document is and how it can be used.

Next, you learned about each of the elements that make up the WSDL document. You also learned how to combine them into a complete WSDL document.

Q&A

Q **What is the purpose of a WSDL document?**

A To completely describe a Web service to a client or potential client.

Q **What part of the WSDL tells what the input parameter looks like?**

A The `types` element provides a description of the input data type. This element is normally formatted using XML schema notation.

Q **What part of the WSDL provides the information about what method calls the Web service accepts and sends back to the client?**

A The `message` element defines individual messages, and the `operation` element combines messages into meaningful transactions.

Q **What part of the WSDL is concerned with the actual establishing of contact between the client and the Web service?**

A The `binding` provides information about the protocol to be used and the transport that the protocol will travel over.

Workshop

The Workshop is designed to help you review what you've learned and begin learning how to put your knowledge into practice.

Quiz

1. What XML grammar is used to specify the message details in a WSDL?
2. Why is the WSDL needed?
3. What are the abstract elements?
4. What are the concrete elements?

Quiz Answers

1. The XML schema is used.
2. To provide capability and connection details to clients and potential clients.
3. `portType`, `operation`, `message`, and `types` are all abstract elements because they describe the Web service in the abstract, but do not provide the details needed for a connection.
4. `service`, `port`, and `binding` are concrete elements because they provide the connection details to a Web service.

Activities

1. Print out the full `customerExample` WSDL document that is located in the code download for this hour. Take a highlighter pen and highlight each part that you understand.
2. For the parts of the `customerExample` WSDL document that you don't understand, reread the section in the hour that covers this topic, and then highlight it.