1

Anatomy of a Web API

In this chapter, you will learn the basic concepts of Web APIs that I use throughout the rest of the book. Web APIs are application programming interfaces that can be called over standard Internet protocols. Many companies are using Web APIs to expose functionality in their existing systems in a platform-neutral manner. Other companies are building applications from the ground up as Web APIs. Web APIs generally allow remote computers on different platforms to talk to each other using methods that were previously very difficult. This chapter will specifically cover the following:

- What a Web API is and how it differs from a Web service
- The current status of Web APIs in the industry
- Calling a Web API using REST (HTTP-GET)
- Calling a Web API using HTTP-POST
- Calling a Web API using HTTP and SOAP

Web APIs versus Web Services

Web APIs are application programming interfaces that are available over the Internet. They are also sometimes referred to as Web services. It is helpful to think of a Web API as a series of Web services, each of which has one or more procedures that can be called using the Internet. An example of a Web API includes the Google API, which makes various search functions available over the Internet for use in third-party applications. Although Web APIs and Web services are separate concepts, it is common and acceptable to refer to the two concepts interchangeably because they are so closely related.

In the most generic sense, a Web service is merely a function or procedure that can be called over the Internet. This generic definition includes Web services that can be called only from specific platforms, such as Windows, and only after installation of certain software on the client. However, in the context of this book, the term *Web service* refers to services that are platform-neutral and so can be called from any platform capable of communicating using standard Internet protocols. These platform-neutral Web services are sometimes referred to as *XML Web services* because XML is typically the data transfer format used for them. XML is a text-based syntax that can be understood by various platforms, as I discuss in a later section. Despite common misconceptions, XML Web services can be called from applications that are not browser-based, such as traditional Windows applications. As long as the application can communicate using Internet protocols, non-Web–based applications can make use of the same functionality.

Despite the great value of Web services, their adoption industry-wide has been slow in coming. The great news is that, as of this writing, Web APIs and Web services have finally been adopted by big players in the industry. Industry leaders such as Google, Amazon, and eBay have embraced the Web services concept and have created Web APIs that enable you to implement their core features in your own applications. This recent movement toward Web APIs is my primary reason for writing this book. In it, you will, of course, explore several leading Web APIs to learn how you can use them in your applications.

As you will see in the following chapters, most of these leading Web APIs require you to obtain a developer token that must be included in each request. Vendors require a developer token in order to control how much you use (and thus not abuse) the service and/or how much they should charge you. Most Web APIs have a free limited-use license or trial period, and some of them require payment for the service.

Web APIs as XML Web Services

As I mentioned previously, Web APIs are typically a related collection of Web services. Most of the Web services available today are based on XML and can, therefore, be called from various platforms. The focus of this book is primarily on XML Web services. Therefore, a brief explanation on XML is appropriate before you learn the basics of calling XML Web services in your programs.

What Is XML?

XML stands for *eXtensible Markup Language*, which is a text-based markup language. XML is similar to HTML in many ways, such as how it describes data by using tags. A very simple example of an XML document is shown here.

```
<?xml version="1.0" encoding="UTF-8"?>
<contact>
<name ContactType="Business">
  <first>John</first>
  <last>Doe</last>
 </name>
 <name ContactType="Personal">
  <first>Jane</first>
  <last>Doe</last>
 </name>
</contact>
```

As you can see from the previous code example, this is a contact record with two contacts, John Doe and Jane Doe. For each contact, a contact type and a first and last name are specified. This XML document can be sent to a mainframe, a Windows computer, a UNIX computer, and a Linux computer. All these computers can read it because it is a text file.

Invoking an XML Web Service

A very important aspect of communicating with a Web service is how you can physically execute a specified function. XML Web services are based on standard Internet protocols and allow you to invoke Web services through communication mechanisms such as *HTTP-GET, HTTP-POST, or SOAP over HTTP*, as I describe in more detail in the following sections. Some Web APIs covered in this book, such as Amazon's Web API, support both the HTTP-GET and SOAP options, although others support only one method. Web service providers can enable all three methods, but they typically disable one or more that they decide are not appropriate or desirable for various reasons. The fact that vendors support different methods is the subject of much debate. You should, however, understand how to use each method so you can work with the various Web APIs that are available. The methods are introduced here, but I will cover them in greater detail in the chapters that follow.

Invoking a Web Service Using REST (HTTP-GET)

REST stands for *Representational State Transfer.* It refers to invoking a Web service using parameters included in a URL. REST uses HTTP-GET to retrieve data and is not typically used for data updates. After a request is processed, REST returns an XML document.

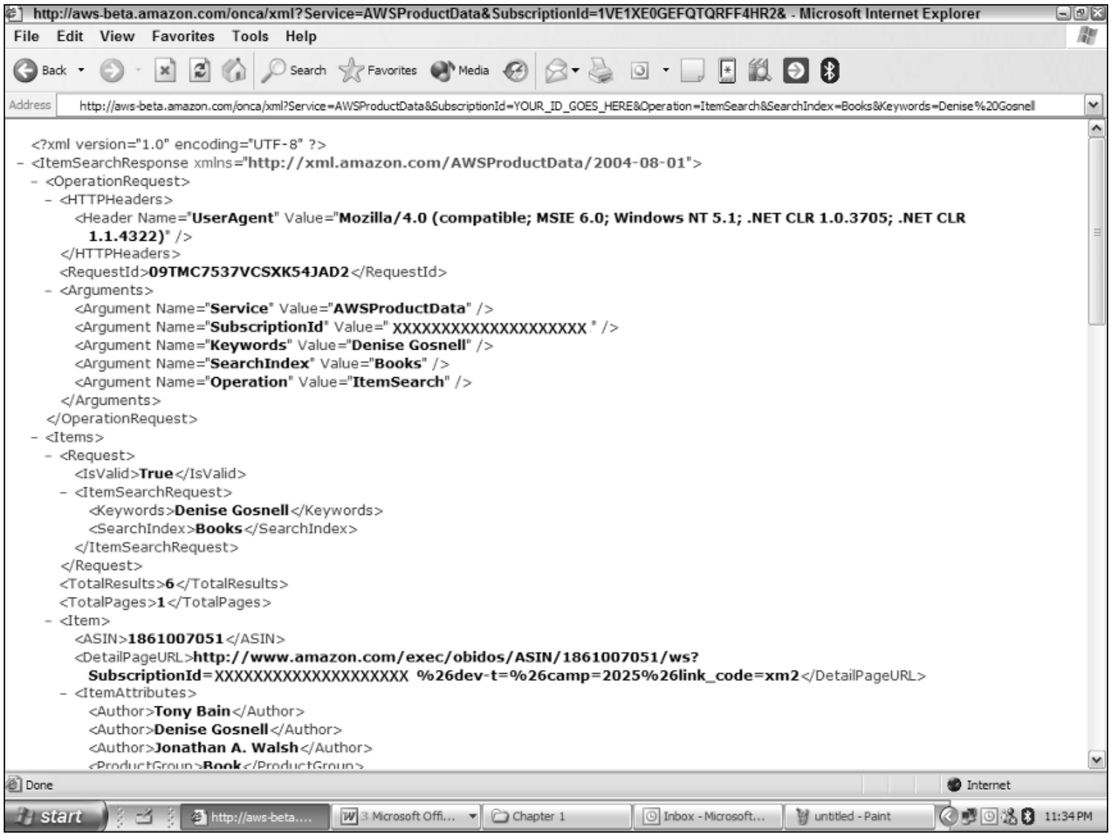
Let's look at an example of calling an Amazon Web service using REST. Suppose you have the following URL:

```
http://aws-
beta.amazon.com/onca/xml?Service=AWSProductData&SubscriptionId=YOUR_ID_GOES_HERE&Op
eration=ItemSearch&SearchIndex=Books&Keywords=Denise%20Gosnell
```

Notice how the first part of the URL contains the traditional domain information. Next, you see the various parameters being passed. These parameters are being passed to an Amazon API (currently version 4.0 beta). The URL includes a parameter for the service being called (AWSProductData), the subscription ID (your developer token), the operation to perform (a search), the search index to use (books), and the keywords to search on (Denise Gosnell). Parameters are separated by ampersands (`&`), and spaces are indicated by `%20`.

As you can see, it is really quite easy to call a Web service using REST, if the Web service supports REST. You can then use your programming language of choice to execute the HTTP-GET command containing the URL and process the XML file that is returned. You will see examples of this in action throughout the book.

It is very easy to test a REST Web service from your Web browser. Pasting the URL listed previously into a Web browser returns results similar to those shown in Figure 1-1.



http://aws-beta.amazon.com/onca/xml?Service=AWSProductData&SubscriptionId=1VE1XE0GEFQTQRFF4HR2& - Microsoft Internet Explorer

Address http://aws-beta.amazon.com/onca/xml?Service=AWSProductData&SubscriptionId=YOUR_ID_GOES_HERE&Operation=ItemSearch&SearchIndex=Books&Keywords=Denise%20Gosnell

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <ItemSearchResponse xmlns="http://xml.amazon.com/AWSProductData/2004-08-01">
  - <OperationRequest>
    - <HTTPHeaders>
        <Header Name="UserAgent" Value="Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705; .NET CLR
          1.1.4322)" />
      </HTTPHeaders>
      <RequestId>09TMC7537VCSXK54JAD2</RequestId>
    - <Arguments>
        <Argument Name="Service" Value="AWSProductData" />
        <Argument Name="SubscriptionId" Value=" XXXXXXXXXXXXXXXXXXXX " />
        <Argument Name="Keywords" Value="Denise Gosnell" />
        <Argument Name="SearchIndex" Value="Books" />
        <Argument Name="Operation" Value="ItemSearch" />
      </Arguments>
    </OperationRequest>
  - <Items>
    - <Request>
        <IsValid>True</IsValid>
      - <ItemSearchRequest>
          <Keywords>Denise Gosnell</Keywords>
          <SearchIndex>Books</SearchIndex>
        </ItemSearchRequest>
      </Request>
      <TotalResults>6</TotalResults>
      <TotalPages>1</TotalPages>
    - <Item>
        <ASIN>1861007051</ASIN>
        <DetailPageURL>http://www.amazon.com/exec/obidos/ASIN/1861007051/ws?
          SubscriptionId=XXXXXXXXXXXXXXXXXXXX %26dev-t=%26camp=2025%26link_code=xm2</DetailPageURL>
      - <ItemAttributes>
          <Author>Tony Bain</Author>
          <Author>Denise Gosnell</Author>
          <Author>Jonathan A. Walsh</Author>
          <ProductGroup>Book</ProductGroup>
```

Figure 1-1

The steps for calling a Web service using REST are summarized as follows:

1. Identify the Web service you would like to call and the parameters it accepts.
2. Formulate the URL containing the parameters.
3. Test the URL from a Web browser to ensure it works correctly.
4. Use your programming language of choice to call the HTTP-GET command with the URL.
5. Receive the results in an XML document and parse the document using an XML parsing method of choice.

Despite the advantages of REST, it also has limitations. For example, you do not want to use REST when you need to transmit sensitive data because you do not want to include sensitive data in a text URL. Also, the URL has a size limit, so REST does not work for all Web APIs. Because REST is performing a HTTP-GET (data retrieval only) operation, it cannot be used to post data. It can still be used to update data as long as the values in the URL do not exceed the maximum allowed length. Let’s now look at another way of invoking a Web service — using HTTP-POST.

Invoking a Web Service Using HTTP-POST

HTTP-POST is very similar to HTTP-GET, but also introduces additional complexities and advantages. When you call a Web service using HTTP-POST, you actually post an XML document that contains the required information for calling the Web service. HTTP-POST then returns an XML document in response. HTTP-POST can be used for data retrieval as well as updates, and it is better for transmitting sensitive information than REST, which sends the information in clear text in a URL.

The steps for calling a Web service using HTTP-POST are summarized as follows:

1. Identify the Web service you would like to call and the parameters it accepts.
2. Formulate an XML document containing the parameters.
3. Use your programming language of choice to call the HTTP-POST command to POST the XML document to the desired Web service.
4. Receive the results in an XML document and parse the document using an XML parsing method of choice.

The following code segment illustrates an example of using HTTP-POST to interact with a Web API.

```
Dim web As New System.Net.WebClient

Dim strXML As String

strXML = "Your XML Document Here"

'add the xml string to the byte array
Dim d As Byte() = System.Text.Encoding.ASCII.GetBytes(strXML)

'call the api and pass the byte array containing the XML string
Dim res As Byte() = web.UploadData("URLForAPI", "POST", d)

'display the results in a message box
MsgBox(System.Text.Encoding.ASCII.GetString(res))
```

As you will see later in this book, eBay supports the HTTP-POST method as well as SOAP, which is described next.

Invoking a Web Service Using SOAP

Another way to call an XML Web service is using *SOAP*. SOAP stands for *Simple Object Access Protocol* and is an XML-based protocol for exchanging structured and type information over the Internet. SOAP, unlike HTTP-GET and HTTP-POST, supports both simple and complex types. Thus, complex types such as datasets, structs, and classes can be used in SOAP communications. SOAP is the primary message format used by the .NET Framework for communicating with XML Web services.

The following is an example of a SOAP document that calls a function named `getUserInfo` on a Web service located on arcweb.esri.com to retrieve user info for the specified token.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:getUserInfo xmlns:m="http://arcweb.esri.com/v2">
      <token xsi:type="xsd:string">MyToken</token>
    </m:getUserInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

As you can see, SOAP documents are a little more difficult to understand than XML documents. But there is good news. Visual Studio .NET and the .NET Framework can handle the SOAP creation for you simply by adding a reference to your project in Visual Studio .NET or by using a compiler tool if you are using just the .NET Framework. Then, all you have to do is declare and call the Web service from your code just as you call other objects. We'll walk through an example later in this chapter so you can see how this works.

Another important concept to understand when working with Web services using SOAP is *Web Services Description Language (WSDL)*. WSDL is an XML-based language that was started as a joint effort by Microsoft and IBM as a way to document what messages the Web service accepts and generates in order to document what procedures you can call and what type of values they will return.

Following are excerpts of a WSDL file that describes the Amazon Web API (version 4.0 beta), which can currently be found at `http://aws-beta.amazon.com/AWSSchemas/AWSProductData/beta/US.wsdl`.

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://xml.amazon.com/AWSProductData/2004-08-01"
targetNamespace="http://xml.amazon.com/AWSProductData/2004-08-01">
<types>
<xs:schema targetNamespace="http://xml.amazon.com/AWSProductData/2004-08-01"
elementFormDefault="qualified">

...[portions omitted]...

<xs:element name="ItemSearch">
<xs:complexType>
<xs:sequence>
  <xs:element name="SubscriptionId" type="xs:string" minOccurs="0" />
  <xs:element name="AssociateTag" type="xs:string" minOccurs="0" />
  <xs:element name="Validate" type="xs:string" minOccurs="0" />
  <xs:element name="XMLEscaping" type="xs:string" minOccurs="0" />
  <xs:element name="Shared" type="tns:ItemSearchRequest" minOccurs="0" />
  <xs:element name="Request" type="tns:ItemSearchRequest" minOccurs="0"
maxOccurs="unbounded" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
```

```
<xs:complexType name="ItemSearchRequest">
<xs:sequence>
  <xs:element name="Actor" type="xs:string" minOccurs="0" />
  <xs:element name="Artist" type="xs:string" minOccurs="0" />
  <xs:element ref="tns:AudienceRating" minOccurs="0" maxOccurs="unbounded" />
  <xs:element name="Author" type="xs:string" minOccurs="0" />
  <xs:element name="Brand" type="xs:string" minOccurs="0" />
  <xs:element name="BrowseNode" type="xs:string" minOccurs="0" />
  <xs:element name="City" type="xs:string" minOccurs="0" />
  <xs:element name="Composer" type="xs:string" minOccurs="0" />
  <xs:element ref="tns:Condition" minOccurs="0" />
  <xs:element name="Conductor" type="xs:string" minOccurs="0" />
  <xs:element name="Cuisine" type="xs:string" minOccurs="0" />
  <xs:element ref="tns:DeliveryMethod" minOccurs="0" />
  <xs:element name="Director" type="xs:string" minOccurs="0" />
  <xs:element name="ISPUPostalCode" type="xs:string" minOccurs="0" />
  <xs:element name="ItemPage" type="xs:positiveInteger" minOccurs="0" />
  <xs:element name="Keywords" type="xs:string" minOccurs="0" />
  <xs:element name="Manufacturer" type="xs:string" minOccurs="0" />
  <xs:element name="MaximumPrice" type="xs:nonNegativeInteger" minOccurs="0" />
  <xs:element name="MerchantId" type="xs:string" minOccurs="0" />
  <xs:element name="MinimumPrice" type="xs:nonNegativeInteger" minOccurs="0" />
  <xs:element name="MusicLabel" type="xs:string" minOccurs="0" />
  <xs:element name="Neighborhood" type="xs:string" minOccurs="0" />
  <xs:element name="Orchestra" type="xs:string" minOccurs="0" />
  <xs:element name="PostalCode" type="xs:string" minOccurs="0" />
  <xs:element name="Power" type="xs:string" minOccurs="0" />
  <xs:element name="Publisher" type="xs:string" minOccurs="0" />
  <xs:element name="ResponseGroup" type="xs:string" minOccurs="0"
maxOccurs="unbounded" />
  <xs:element name="SearchIndex" type="xs:string" minOccurs="0" />
  <xs:element name="Sort" type="xs:string" minOccurs="0" />
  <xs:element name="State" type="xs:string" minOccurs="0" />
  <xs:element name="TextStream" type="xs:string" minOccurs="0" />
  <xs:element name="Title" type="xs:string" minOccurs="0" />
  </xs:sequence>
  </xs:complexType>

...[portions omitted]...

<service name="AWSProductData">
<port name="AWSProductDataPort" binding="tns:AWSProductDataBinding">
  <soap:address location="http://aws-
beta.amazon.com/onca/soap?Service=AWSProductData" />
  </port>
  </service>
  </definitions>
```

Notice that details about the `ItemSearch` and `ItemSearchRequest` elements are described. These are functions that can be called in the `AWSProductData` Web service of the Amazon API. The elements that you see for these functions are the parameters that the functions can accept.

The general steps for calling a Web service using SOAP from the .NET Framework (without using Visual Studio .NET) include the following:

1. Using a text editor (such as Notepad), create all or part of the .NET program where you want to use the Web service.
2. Create a *Web Service Proxy Class DLL* manually using the *WSDL.EXE* command line tool.
3. Revise the .NET program to import the namespace of the Web Service Proxy Class DLL created in Step 2.
4. Revise the .NET program to include the lines of code that call one or more functions of the Web service.

The general steps for calling a Web service using SOAP from Visual Studio .NET include the following:

1. Create a new project in Visual Studio .NET.
2. Add a Web Reference to point to the Web service you wish to call.
3. Add code in your program to create an instance of the class representing the Web service, and then call the appropriate methods in the Web service.

Walkthrough Example — Calling a Web Service Using SOAP from Visual Studio .NET

Now that you have been introduced to some basic concepts, let's walk through a step-by-step example of using SOAP and WSDL to call a Web service from Visual Studio .NET. In this example, you call the same Amazon Web service with the same search criteria that you saw in the REST examples earlier in this chapter. Because this example uses SOAP and Visual Studio.Net, the steps are quite different to call that same Web service.

Note that you need to obtain your own Amazon Web API Associate ID/Developer ID if you want to run the following example yourself. You can find more details about how to obtain an Amazon Associate ID in Chapter 4.

1. Open Visual Studio .NET.
2. Select File ➪ New ➪ Project, and select Windows Application as the type of project.
3. Specify `TestAmazonWebService` or another suitable name for the Name and change the project location if desired. Then click OK.
4. Select Project ➪ Add Web Reference, as shown in Figure 1-2.
5. A screen like the one shown in Figure 1-3 is then displayed to allow you to specify the Web service to which you want to add a reference.

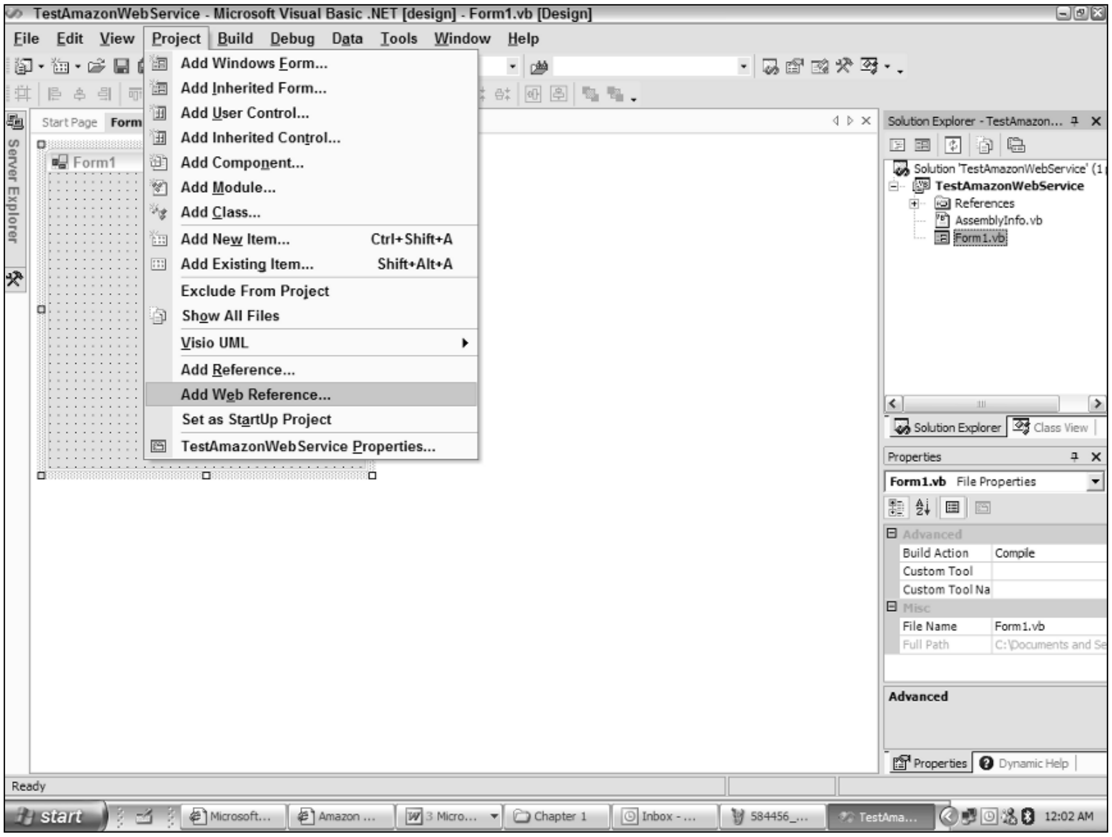


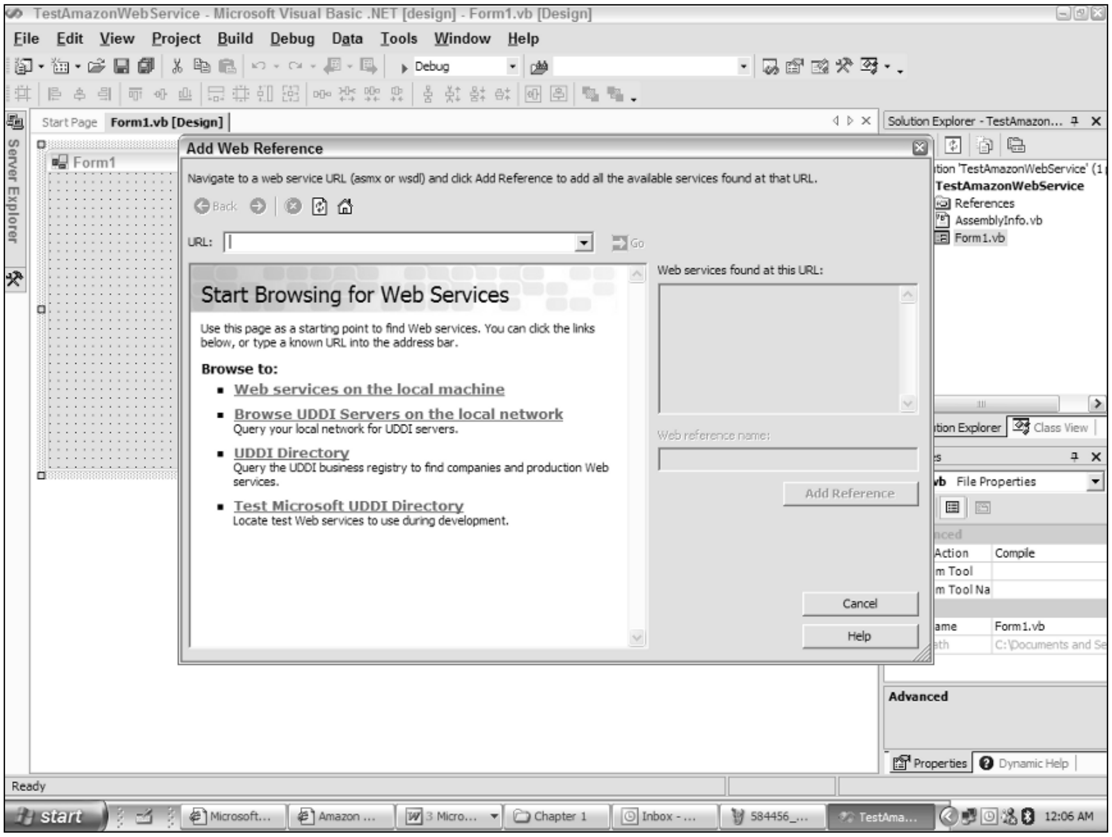
Figure 1-2

6. In the URL field, type or paste the URL where the WSDL file of the API you want to work with is located. Alternatively, you can search for Web services using the links indicated. In this example, specify the location of the Amazon API (version 4.0 or higher), which is currently located at: `http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl`

7. After specifying the URL for the location of the WSDL file, click Go. A list of the methods available for that Web service is then displayed, as shown in Figure 1-4.

8. Change the Web reference name to `Amazon` and click the Add Reference button indicated on Figure 1-4.

9. You will notice that multiple references are added to the project, as shown in Figure 1-5, along with a Web reference called `Amazon`. This name is the Web reference name that was specified on the Add Reference dialog box.



Add Web Reference

Navigate to a web service URL (asmx or wsdl) and click Add Reference to add all the available services found at that URL.

Back

URL:

Go

Start Browsing for Web Services

Use this page as a starting point to find Web services. You can click the links below, or type a known URL into the address bar.

Browse to:

- Web services on the local machine
- Browse UDDI Servers on the local network
  Query your local network for UDDI servers.
- UDDI Directory
  Query the UDDI business registry to find companies and production Web services.
- Test Microsoft UDDI Directory
  Locate test Web services to use during development.

Web services found at this URL:

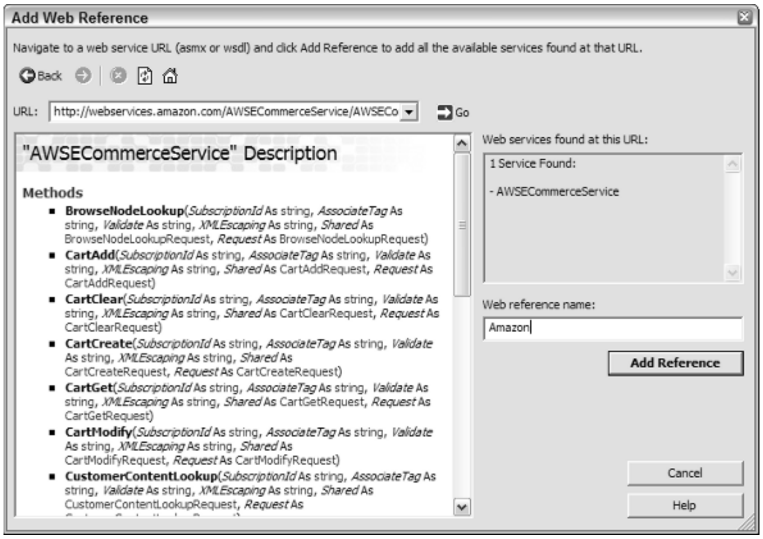
Web reference name:

Add Reference

Cancel

Help

Figure 1-3



Add Web Reference

Navigate to a web service URL (asmx or wsdl) and click Add Reference to add all the available services found at that URL.

Back

URL: http://webservices.amazon.com/AWSECommerceService/AWSECo

Go

"AWSECommerceService" Description

Methods

- **BrowseNodeLookup**(*SubscriptionId* As string, *AssociateTag* As string, *Validate* As string, *XMLEscaping* As string, *Shared* As BrowseNodeLookupRequest, *Request* As BrowseNodeLookupRequest)
- **CartAdd**(*SubscriptionId* As string, *AssociateTag* As string, *Validate* As string, *XMLEscaping* As string, *Shared* As CartAddRequest, *Request* As CartAddRequest)
- **CartClear**(*SubscriptionId* As string, *AssociateTag* As string, *Validate* As string, *XMLEscaping* As string, *Shared* As CartClearRequest, *Request* As CartClearRequest)
- **CartCreate**(*SubscriptionId* As string, *AssociateTag* As string, *Validate* As string, *XMLEscaping* As string, *Shared* As CartCreateRequest, *Request* As CartCreateRequest)
- **CartGet**(*SubscriptionId* As string, *AssociateTag* As string, *Validate* As string, *XMLEscaping* As string, *Shared* As CartGetRequest, *Request* As CartGetRequest)
- **CartModify**(*SubscriptionId* As string, *AssociateTag* As string, *Validate* As string, *XMLEscaping* As string, *Shared* As CartModifyRequest, *Request* As CartModifyRequest)
- **CustomerContentLookup**(*SubscriptionId* As string, *AssociateTag* As string, *Validate* As string, *XMLEscaping* As string, *Shared* As CustomerContentLookupRequest, *Request* As

Web services found at this URL:

1 Service Found:

- AWSECommerceService

Web reference name: Amazon

Add Reference

Cancel

Help

Figure 1-4

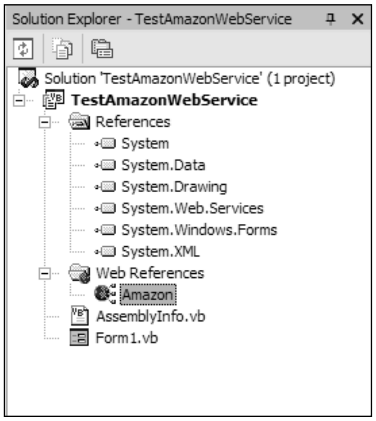


Figure 1-5

10. Add the following code to the `Form1_Load` event of Form1.

```
      Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
           System.EventArgs) Handles MyBase.Load

          Dim AmazonProductData As New _
              TestAmazonWebService.Amazon.AWSECommerceService
          Dim AmazonSearch As New TestAmazonWebService.Amazon.ItemSearch
          Dim AmazonResponse As New _
              TestAmazonWebService.Amazon.ItemSearchResponse
          Dim AmazonRequest(1) As _
              TestAmazonWebService.Amazon.ItemSearchRequest

          'Developer/Subscription Code
          AmazonSearch.SubscriptionId = "YOUR ID GOES HERE"

          'We are only making one request, not batching multiple requests
          'Thus element 0 of the array is all we need to assign and work with
          AmazonRequest(0) = New _
              TestAmazonWebService.Amazon.ItemSearchRequest
          AmazonRequest(0).SearchIndex = "Books"
          AmazonRequest(0).Keywords = "Denise Gosnell"

          'assign the search object to request object with the assigned parameters
          AmazonSearch.Request = AmazonRequest

          'run the search and populate the response
          AmazonResponse = AmazonProductData.ItemSearch(AmazonSearch)

          Dim item As New TestAmazonWebService.Amazon.Item
          Dim strOutput As String

          strOutput = "Search results for keyword(s): " & AmazonRequest(0).Keywords &
  " in " & AmazonRequest(0).SearchIndex & ":" & vbCrLf & vbCrLf
```

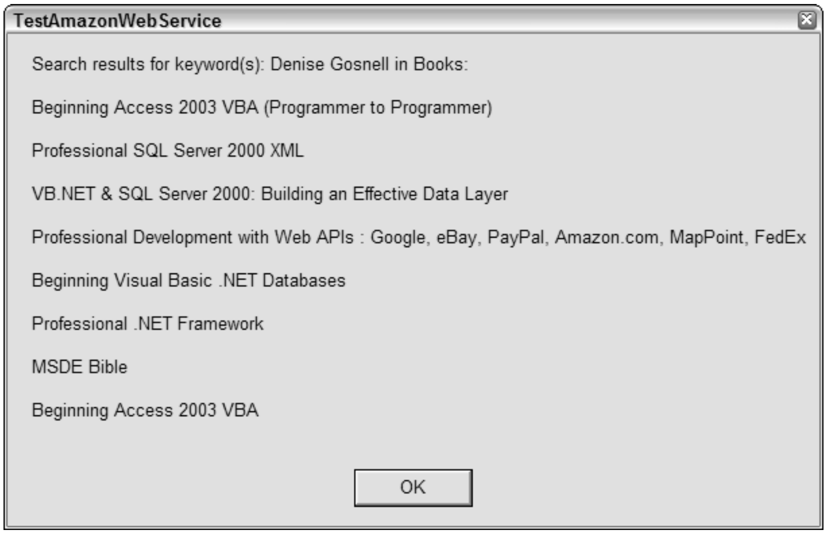
```
            'loop through the results
            For Each item In AmazonResponse.Items(0).Item
                strOutput = strOutput & item.ItemAttributes.Title & vbCrLf & vbCrLf
            Next

            'display the results in a message box
            MsgBox(strOutput)

        End Sub
```

Note that you must specify your own Associate ID for the Associate ID value in the previous code or the example will not run for you.

11. Run the program by pressing F5 or by selecting Debug ➪ Start. You should see results similar to those shown in Figure 1-6.



```
Search results for keyword(s): Denise Gosnell in Books:

Beginning Access 2003 VBA (Programmer to Programmer)

Professional SQL Server 2000 XML

VB.NET & SQL Server 2000: Building an Effective Data Layer

Professional Development with Web APIs : Google, eBay, PayPal, Amazon.com, MapPoint, FedEx

Beginning Visual Basic .NET Databases

Professional .NET Framework

MSDE Bible

Beginning Access 2003 VBA
```

Figure 1-6

Also note that if you receive an error message when running the program stating that CustomerReviews1 cannot be reflected, please consult Chapter 4 for specific instructions on how to resolve the error.

Let's briefly review what just happened. You added a Web reference to the Amazon Web service (you pointed to the Amazon WSDL file). Visual Studio .NET used the WSDL file to create the objects that enable you to use the IntelliSense feature as you type and to refer to the Web services in your code just as you do other objects.

You then added code to the `Form_Load` event of Form1 to call one of the available Amazon Web services and to display a message box illustrating part of the search results. Review the VB code in more detail to see exactly how that works.

First, you declared various Amazon Web service objects that are required in order to execute an item search.

```
        Dim AmazonProductData As New TestAmazonWebService.Amazon.
AWSECommerceService
        Dim AmazonSearch As New TestAmazonWebService.Amazon.ItemSearch
        Dim AmazonResponse As New TestAmazonWebService.Amazon.ItemSearchResponse
        Dim AmazonRequest(1) As TestAmazonWebService.Amazon.ItemSearchRequest
```

You then supplied an Amazon Subscription ID.

```
        'Developer/Subscription Code
        AmazonSearch.SubscriptionId = "YOUR ID GOES HERE"
```

Because only one request is being made to the Web service, as opposed to a batch with multiple requests, element 0 of the array is used and assigned the search parameter values.

```
        'We are only making one request, not batching multiple requests
        'Thus element 0 of the array is all we need to assign and work with
        AmazonRequest(0) = New TestAmazonWebService.Amazon.ItemSearchRequest
        AmazonRequest(0).SearchIndex = "Books"
        AmazonRequest(0).Keywords = "Denise Gosnell"
```

You then assign the search object to the request object so that the search object will have the required parameters.

```
        'assign the search object to the request object with the assigned
parameters
        AmazonSearch.Request = AmazonRequest
```

Next, you called the `ItemSearch` method and assigned the results to the `AmazonResponse` variable.

```
        'run the search and populate the response
        AmazonResponse = AmazonProductData.ItemSearch(AmazonSearch)
```

Finally, you looped through the items in the response to build an output string for display to the user.

```
        Dim item As New TestAmazonWebService.Amazon.Item
        Dim strOutput As String

        strOutput = "Search results for keyword(s): " & AmazonRequest(0).Keywords &
" in " & AmazonRequest(0).SearchIndex & ":" & vbCrLf & vbCrLf

        'loop through the results
        For Each item In AmazonResponse.Items(0).Item
            strOutput = strOutput & item.ItemAttributes.Title & vbCrLf & vbCrLf
        Next

        'display the results in a message box
        MsgBox(strOutput)

    End Sub
```

This is just one example of the many ways you can use the Amazon Web service. For more information on the Amazon Web API, consult Chapter 4, which is dedicated exclusively to the Amazon Web API.

Summary

In this chapter, you explored the world of Web APIs and Web services. You learned about the various ways to call a Web service, such as using HTTP-GET, HTTP-POST, and SOAP.

Web services are growing in popularity every day. Now that major players such as Google, Amazon, eBay, and others have released Web APIs, Web services will pick up even more steam. By using the Web service APIs of these leading vendors, you can enhance your own applications tremendously. The remaining chapters are dedicated to illustrating several different Web service APIs in detail and to providing you with some real-world examples of using those APIs alone and in combination from mobile devices, Microsoft Office, and in various other ways.

Let's turn to the next chapter, which focuses on the Google API.