



United States Postal Service®



Web Tool Kit User's Guide

A Technical Guide to

Address Informational

Application Programming Interfaces

Address Standardization

ZIP Code Lookup

City/State Lookup



Before implementing this API, the *Administrative Guide for Application Programming Interfaces* must be read.

Version 2.2 (06/09/03)

To Our Customers

In the e-mail that accompanied this guide you received a password and user ID that will allow you to begin sending calls to the “test server” when you are ready. Any additional documentation or contact with you will be made through the contact person indicated on the registration form.

If you require technical support, contact the USPS Internet Customer Care Center (ICCC). This office is manned from 7:00AM to 11:00PM EST.

E-mail: icustomer@usps.com

Telephone: 1-800-344-7779 (7:00AM to 11:00PM EST)

USPS Customer Commitment

The United States Postal Service fully understands the importance of providing information and service anytime day or night to your Internet and e-commerce customers. For that reason, the USPS is committed to providing 7 x 24 service from our API servers, 365 days a year.

Thank you for helping the U.S. Postal Service provide new Internet services to our shipping customers.

Internet Shipping Solutions Team
U.S. Postal Service
475 L'Enfant Plaza, SW
Washington, DC 20260-2464

Trademarks

Express Mail, First-Class Mail, Global Priority Mail, Parcel Post, Parcel Select, Priority Mail, and ZIP + 4 are registered trademarks of the U.S. Postal Service.

Delivery Confirmation, Global Express Guaranteed, Global Express Mail, GXG, International Parcel Post, Priority Mail Global Guaranteed, Signature Confirmation, and ZIP Code are trademarks of the U.S. Postal Service.

Microsoft, Visual Basic, and Word are registered trademarks of Microsoft Corporation.

Adobe Acrobat is a trademark of Adobe Systems Incorporated.

©Copyright 2002 United States Postal Service

Table of Contents

Introduction to the Address Informational APIs	1
User ID and Password Restrictions	2
Transaction Procedures for Address Standardization API	3
Technical Steps	3
Step 1: Build the XML Request	3
"Cannd" Test Requests	3
Valid Test Requests	3
Pre-Defined Error Requests	4
"Live" Request.....	5
Visual Basic Request.....	6
Steps 2 & 3: Make the Internet Connection and Send the XML Request	7
Using HTTP Connection DLL	7
Using WinInet.....	8
Step 4: Unpack the XML Response	9
Types of Responses	9
Using Visual Basic	9
Errors	10
Output	11
"Cannd" Test Responses	11
"Live" Responses	14
XML Output Example	14
Transaction Procedures for ZIP Code Lookup API.....	15
Technical Steps	15
Step 1: Build the XML Request	15
"Cannd" Test Requests	15
Valid Test Requests	15
Pre-Defined Error Requests	16
"Live" Request.....	17
Visual Basic Request.....	18
Steps 2 & 3: Make the Internet Connection and Send the XML Request	19
Using HTTP Connection DLL	20
Using WinInet.....	20
Step 4: Unpack the XML Response	21
Types of Responses	21
Using Visual Basic	21
Errors	23
Output	24
"Cannd" Test Responses	24
"Live" Responses	26
XML Output Example	26
Transaction Procedures for City/State Lookup API.....	28
Technical Steps	28
Step 1: Build the XML Request	28

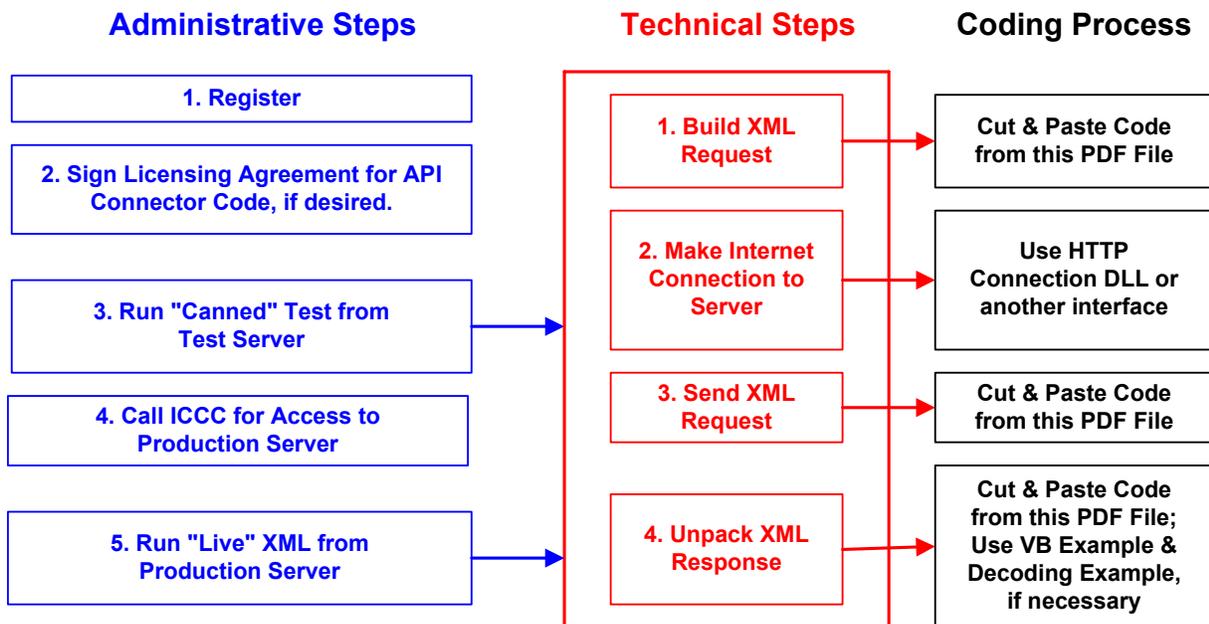
“Canned” Test Requests 28
 Valid Test Requests 28
 Pre-Defined Error Request 29
“Live” Request 29
 Visual Basic Request 30
Steps 2 & 3: Make the Internet Connection and Send the XML Request 31
 Using HTTP Connection DLL 31
 Using WinInet 31
Step 4: Unpack the XML Response 32
 Types of Responses 33
 Using Visual Basic 33
 Errors 35
Output 35
 “Canned” Test Responses 36
 “Live” Responses 38
 XML Output Example 38

Introduction to the Address Informational APIs

There are three Address Informational APIs provided in this guide:

1. **Address Standardization API.** This API corrects errors in street addresses, including abbreviations and missing information, and supplies ZIP Codes and ZIP Codes + 4. It supports up to five lookups per transaction. By eliminating address errors, you will improve overall package delivery service.
2. **ZIP Code Lookup API.** The ZIP Code Lookup API returns the ZIP Code and ZIP Code + 4 corresponding to the given address, city, and state (use USPS state abbreviations). The ZIP Code Lookup API processes up to five lookups per request.
3. **City/State Lookup API.** The City/State Lookup API returns the city and state corresponding to the given ZIP Code. The City/State Lookup API processes up to five lookups per request.

As shown below, implementing USPS Shipping APIs requires a series of *Administrative Steps*. The *Administrative Guide for APIs*, sent to you with this document, provides necessary information and procedures prior to installation. The illustration also shows the *Technical Steps* required to run XML transactions to either the test server or the production server, as well as the *Coding Process* to be followed for each *Technical Step*. This document provides step-by-step instructions for both the Technical Steps and Coding Process illustrated below.



Implementing these APIs requires experienced programmers who are familiar with Internet and web site development tools and techniques. Before implementing this API, the Administrative Guide for Application Programming Interfaces must be read.

User ID and Password Restrictions

The user ID and password that you have received is only for the use of you or your company in accordance with the Terms and Conditions of Use to which you agreed during the registration process. *This user ID and password is not to be shared with others outside your organization, nor is it to be packaged, distributed, or sold to any other person or entity.* Please refer to the Terms and Conditions of Use Agreement for additional restrictions on the use of your user ID and password, as well as this document and the APIs contained herein.

The documentation and sample code contained in the *Web Tool Kit User Guide* series may be reused and/or distributed to your customers or affiliates to generate awareness, encourage web tool use, or provide ease-of-use. However, it is your responsibility to ensure that your customers do not use your password and user ID. Direct them to www.uspsprioritymail.com so that they can register, agree to the Terms and Conditions of Use agreement, and receive their own unique password and user ID.

Warning: If the U.S. Postal Service discovers use of the same user ID and password from more than one web site, all users will be subject to immediate loss of access to the USPS server and termination of the licenses granted under the Terms and Conditions of Use.

For more information regarding the USPS Web Tool Kit password and user ID policy, or for questions regarding the distribution of documentation, send e-mail to icustomer@usps.com.

Transaction Procedures for Address Standardization API

The illustration below shows the transactional flow of information to and from the USPS Address Standardization API server.

Address Standardization API Server



INPUTS

(via XML Request from Customer to USPS)

Customer Name & Address

SERVER TASKS

Looks Up in Address Management System
Gets Correct Address
Builds XML Response

OUTPUTS

(via XML Response from USPS to Customer)

Corrected Address

Technical Steps

Step 1: Build the XML Request

“Canned” Test Requests

For testing purposes, the only values in the test code in this section that you should change are the “userid” and “password.” Enter the user ID and password you received in the registration e-mail. *All remaining code in the test scripts provided below must remain unchanged.*

All of the test script code contained in this document can be cut and pasted for your use in testing the software. To copy the test script code from this PDF file, click on the icon for “Text Selector” and highlight the code. (The icon will look like



or



depending on your version of Adobe Acrobat.) You can then copy the code and paste it into your test document.

Valid Test Requests

There are four valid requests included in this procedure:

Valid Request #1

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<AddressValidateRequest%
20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="0"><Address1></Address1>
<Address2>6406 Ivy Lane</Address2><City>Greenbelt</City><State>MD</State>
<Zip5></Zip5><Zip4></Zip4></Address></AddressValidateRequest>
```

Valid Request #2

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<AddressValidateRequest%
20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="1"><Address1></Address1>
<Address2>8 Wildwood Drive</Address2><City>Old Lyme</City><State>
CT</State><Zip5>06371</Zip5><Zip4></Zip4></Address></AddressValidateRequest>
```

Valid Request #3

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<AddressValidateRequest%
20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="2"><Address1></Address1>
<Address2>4411 Romlon Street</Address2><City>Beltsville</City><State>MD
</State><Zip5></Zip5><Zip4></Zip4></Address></AddressValidateRequest>
```

Valid Request #4

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<AddressValidateRequest%
20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="3"><Address1></Address1>
<Address2>3527 Sharonwood Road Apt. 3C</Address2><City>Laurel</City><State>
MD</State><Zip5></Zip5><Zip4></Zip4></Address></AddressValidateRequest>
```

Pre-Defined Error Requests

There are four pre-defined errors included for this procedure. Be sure to note the request numbers so you can match up the responses you will receive as provided in the *“Canned” Test Responses* section.

Pre-defined Error Request #1: *“The Address Could Not Be Found”*

For testing purposes, this error will occur when the state input is “DE.”

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<AddressValidateRequest%
20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="0"><Address1></Address1>
<Address2>3527 Sharonwood Road Apt. 3C</Address2><City>Wilmington</City>
<State>DE</State><Zip5></Zip5><Zip4></Zip4></Address></AddressValidateRequest
>
```

Pre-defined Error Request #2:

“Multiple Addresses Were Found and There Is No Default Available”

For testing purposes, this error will occur when the state input is “DC.”

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<AddressValidateRequest%
20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="0"><Address1></Address1>
<Address2>1600 Pennsylvania Avenue</Address2><City>Washington</City><State>
DC</State><Zip5></Zip5><Zip4></Zip4></Address></AddressValidateRequest>
```

Pre-defined Error Request #3: *“The State is Invalid”*

This error will occur when the state input is “ZZ.”

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<AddressValidateRequest%
20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="0"><Address1></Address1>
<Address2>123 Main Street</Address2><City>Washington</City><State>ZZ</State>
<Zip5></Zip5><Zip4></Zip4></Address></AddressValidateRequest>
```

Pre-defined Error Request #4: “The City Is Invalid”

This error will occur when the state input is “NJ.”

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<AddressValidateRequest%
20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="0"><Address1></Address1>
<Address2>123 Main Street</Address2><City>Trenton</City><State>NJ</State>
<Zip5></Zip5><Zip4></Zip4></Address></AddressValidateRequest>
```

“Live” Request

Refer to the “Canned” Test Requests section above for instructions on how to cut and paste the sample code from this PDF file.

Remember that you are provided with a different server name to send “live” requests.

When building the XML request, pay particular attention to the *order and case* for tags.

The table below presents the *required* XML input tags for generating “Live” requests and the restrictions on the values allowed. An error message will be returned if the tag does not contain a value or if an incorrect value is entered. Also, be aware of the maximum character amounts allowed for some tags. If the user enters more than those amounts, an error will not be generated. **The API will simply pass in the characters up to the maximum amount allowed and disregard the rest.** This is important since the resulting value could prevent delivery.



Developers: For sample code utilizing Perl and ASP, refer to the Domestic Rates Calculator API and Track/Confirm API user’s guides.

Input	XML Tag	Values Allowed
Type of Request	<AddressValidateRequest...	Input tag exactly as presented.
User ID	...USERID="userid"...	Use user ID provided with registration.
Password	...PASSWORD="password">	Use password provided with registration.
Address Verification Number	<Address ID='#'>	Up to five address verifications can be included per transaction.
Name of Firm	<FirmName>	Providing the firm name tag is optional. Maximum characters allowed: 38
Address Line 1	<Address1>	Address Line 1 is used to provide an apartment or suite number, if applicable. If not applicable, include the open and close tag with no input. Maximum characters allowed:38
Address Line 2	<Address2>	Street address. Maximum characters allowed: 38
City	<City>	Maximum characters allowed: 15
State	<State>	Maximum characters allowed: 2
ZIP Code	<Zip5>	Input tag exactly as presented, not all caps. Maximum characters allowed: 5
ZIP Code + 4	<Zip4>	Input tag exactly as presented, not all caps. Maximum characters allowed: 4

The “Live” XML request should be in the form:

```
<AddressValidateRequest USERID="xxxxxxx" PASSWORD="xxxxxxx">
  <Address ID="0">
    <FirmName>XYZ Corp.</FirmName>
    <Address1></Address1>
    <Address2>6406 Ivy </Address2>
    <City>Greenbelt</City>
    <State>MD</State>
    <Zip5></Zip5>
    <Zip4></Zip4>
  </Address>
</AddressValidateRequest>
```

Visual Basic Request

Using the Microsoft XML object model in Visual Basic, such a request can be built as shown below. In this code sample, the data needed to build the XML is obtained from a form. The <ServiceType> element is obtained from an option button control and the <ImageType> is from a combo box control. All other fields are obtained from text box controls.

```
Dim xmlDoc As New XmlDocument
Dim RequestLevel As IXMLDOMElement
Dim AddressLevel As IXMLDOMElement
Dim AddressElementLevel As IXMLDOMElement
Dim t As Variant
Dim i As Integer

Set RequestLevel = xmlDoc.createElement("AddressValidateRequest")
RequestLevel.setAttribute "USERID", "SOLClient"
RequestLevel.setAttribute "PASSWORD", "mypassword"

For i = 0 To ?

  Set AddressLevel = xmlDoc.createElement("Address")
  AddressLevel.setAttribute "ID", i
  Set AddressElementLevel = xmlDoc.createElement("FirmName")
  Set t = xmlDoc.createTextNode(txtFirm.Text)
  AddressElementLevel.appendChild (t)
  Call AddressLevel.appendChild(AddressElementLevel)

  Set AddressElementLevel = xmlDoc.createElement("Address1")
  Set t = xmlDoc.createTextNode(txtAddress1.Text)
  AddressElementLevel.appendChild (t)
  Call AddressLevel.appendChild(AddressElementLevel)

  Set AddressElementLevel = xmlDoc.createElement("Address2")
  Set t = xmlDoc.createTextNode(txtAddress2.Text)
  AddressElementLevel.appendChild (t)
  Call AddressLevel.appendChild(AddressElementLevel)

  Set AddressElementLevel = xmlDoc.createElement("City")
  Set t = xmlDoc.createTextNode(txtCity.Text)
  AddressElementLevel.appendChild (t)
  Call AddressLevel.appendChild(AddressElementLevel)

  Set AddressElementLevel = xmlDoc.createElement("State")
  Set t = xmlDoc.createTextNode(cmbState.Text)
  AddressElementLevel.appendChild (t)
```

```
Call AddressLevel.appendChild(AddressElementLevel)

Set AddressElementLevel = xmlDoc.createElement("Zip5")
Set t = xmlDoc.createTextNode(txtZip5.Text)
AddressElementLevel.appendChild (t)
Call AddressLevel.appendChild(AddressElementLevel)

Set AddressElementLevel = xmlDoc.createElement("Zip4")
Set t = xmlDoc.createTextNode(txtZip4.Text)
AddressElementLevel.appendChild (t)
Call AddressLevel.appendChild(AddressElementLevel)

Call RequestLevel.appendChild(AddressLevel)

Next i

Call xmlDoc.appendChild(RequestLevel)
```

Steps 2 & 3: Make the Internet Connection and Send the XML Request

These two steps are presented together to simplify things. The two steps actually involve four separate functions:

1. Making the connection to the USPS Shipping API server (test or production server)
2. Sending the request (whether Visual Basic, Perl, ASP, or any other language)
3. Receiving the response from the API server
4. Closing the Internet connection

These steps are identical for sending “Canned” test requests or “Live” requests. **Remember, however, that you are provided with a different server name to send “live” requests.**

This section provides two samples to make the Internet connection. This is not an all-inclusive list. It simply represents the most common and easiest ways to make the Internet connection.

- Using the USPS-supplied HTTP Connection DLL

The HTTP Connection DLL is recommended for NT systems. This software, created specifically for the USPS API implementation, provides e-tailers with a thread-safe sockets interface to submit XML requests and receive XML responses from the API server.

- Using Microsoft's WinInet

Although you can use the WinInet DLL to make the connection to the API server, it is not recommended for server applications due to limitations in the DLL. It is recommended that you either use the USPS-supplied HTTP Connection DLL or write your own sockets interface that can be used to make multiple connections and will remain thread-safe.

Using HTTP Connection DLL

To obtain this code you must submit a Licensing Agreement. See the *Administrative Guide for APIs* for the agreement.

Using WinInet

This sample code shows how to use Microsoft's WinInet dll to make the Internet connection, using either the "GET" or "POST" (necessary for requests over 2K in size) methods. XMLSTRING represents the URL-encoded XML request and SERVERNAME indicates the name of the USPS web site to which you are connecting. For more information on the Microsoft WinInet product, go to http://msdn.microsoft.com/library/techart/msdn_vbhttp.htm.

Although you can use the WinInet dll to make the connection to the API server, *it is not recommended for server applications due to limitations in the dll*. It is recommended that you write a sockets interface that can be used to make multiple connections and will remain thread-safe.

```
Dim hOpen As Long, hConnection As Long, hFile As Long, numread As Long
Dim File As String, xml As String, sHeader As String, htmlFile As String, tmp
As String * 2048
Dim bDoLoop As Boolean

File = "/ShippingAPI.dll?"
xml = "API=Verify&XML=" & XMLSTRING

hOpen = InternetOpen("", 1, vbNullString, vbNullString, 0)

hConnection = InternetConnect(hOpen, SERVERNAME, 0, _
    "", "", 3, 0, 0)

'.....
'get
'File = File & xml
'hFile = HttpOpenRequest(hConnection, "GET", File, "HTTP/1.0", vbNullString,
0, 0, 0)
'OR
'.....

'.....
' post
hFile = HttpOpenRequest(hConnection, "POST", File, "HTTP/1.0", vbNullString,
0, 0, 0)

sHeader = "Content-Type: application/x-www-form-urlencoded" _
    & vbCrLf

Call HttpAddRequestHeaders(hFile, _
    sHeader, Len(sHeader), 0)
'.....

bDoLoop = HttpSendRequest(hFile, vbNullString, 0, xml, Len(xml))

bDoLoop = True
While bDoLoop
    tmp = vbNullString
    bDoLoop = InternetReadFile(hFile, tmp, Len(tmp), numread)
    If Not bDoLoop Then
        Exit Sub
    Else
```

```
        htmlFile = htmlFile & Left$(tmp, numread)
        If Not CBool(numread) Then bDoLoop = False
    End If
Wend
```

```
If hFile <> 0 Then InternetCloseHandle (hFile)
If hConnection <> 0 Then InternetCloseHandle (hConnection)
If hOpen <> 0 Then InternetCloseHandle (hOpen)
```

Step 4: Unpack the XML Response

This step is identical for unpacking “Canned” test responses or “Live” responses.

Types of Responses

When the USPS Shipping API returns a response, it will either return a successful response document or an error document. Anytime you receive a response, you should check to see if the document is <Error>. Refer to the *Errors* section.

Using Visual Basic

Using the Microsoft XML object model in Visual Basic, such responses can be unpacked as follows:

```
Dim xmlDoc As New XmlDocument
Dim nodeList As IXMLDOMNodeList
Dim n As IXMLDOMNode, e As IXMLDOMNode, t As IXMLDOMNode
Dim i As Integer, j As Integer, k As Integer

xmlDoc.validateOnParse = False
xmlDoc.loadXML (xmlStr) 'Response
If xmlDoc.documentElement.nodeName="Error" then 'Top-level Error
Set nodeList = xmlDoc.getElementsByTagName("Error")
Set n = nodeList.Item(0)
    For i = 0 To n.childNodes.length - 1
        Set e = n.childNodes.Item(i)
        Select Case e.nodeName
            Case "Source"
            Case "Number"
            Case "Description"
                lblAddressMessage.Caption =
                    e.firstChild.nodeValue
            Case "HelpFile"
            Case "HelpContext"
        End Select
    Next i
Else 'no Top-level Error
    Set nodeList = xmlDoc.getElementsByTagName("Address")
    For i = 0 To nodeList.length - 1
        Set n = nodeList.Item(i)
        For j = 0 To n.childNodes.length - 1
            Set e = n.childNodes.Item(j)
            If e.nodeName = "Error" Then 'Lower-level error
                For k = 0 To e.childNodes.length - 1
```

```
        Set t = e.childNodes.Item(k)
        Select Case t.nodeName
            Case "Source"
            Case "Number"
            Case "Description"
                lblAddressMessage(i).Caption =
                    t.firstChild.nodeValue
            Case "HelpFile"
            Case "HelpContext"
        End Select
    Next k
Else 'No error in Package
    Select Case e.nodeName
        Case "Address1"
            txtAddress1(i).Text =
                e.firstChild.nodeValue
        Case "Address2"
            txtAddress2(i).Text =
                e.firstChild.nodeValue
        Case "City"
            txtCity(i).Text =
                e.firstChild.nodeValue
        Case "State"
            txtState(i).Text =
                e.firstChild.nodeValue
        Case "Zip5"
            txtZip5(i).Text =
                e.firstChild.nodeValue
        Case "Zip4"
            txtZip4(i).Text =
                e.firstChild.nodeValue
        Case "ReturnText"
            txtReturntext(i).Text =
                e.firstChild.nodeValue
    End Select
End If
Next j
Next i
End If
```

Errors

Error conditions are handled at the main XML document level. For APIs that can handle multiple transactions, the error conditions for requests for multiple responses to be returned together are handled at the response level. For example: an API developer sends a request for rates for two packages. If the addresses are non-existent, an “Error document” is returned to the user. On the other hand, if the address for the first package is acceptable but not the second, the response document contains the information for the first address, but under the XML tag for the second address there is an error tag.

Error documents follow the Visual Basic error standards and have following format:

```

<Error>
  <Number></Number>
  <Source></Source>
  <Description></Description>
  <HelpFile></HelpFile>
  <HelpContext></HelpContext>
</Error>

```

where:

- Number = the error number generated by the API server
- Source = the component and interface that generated the error on the API server
- Description = the error description
- HelpFile = [reserved]
- HelpContext = [reserved]

Errors that are further down in the hierarchy also follow the above format.

Output

After following Technical Step 4 and unpacking the XML response, you will have the output from your request. This section describes the different outputs resulting from “Canned” test requests and “Live” requests. Both types of requests result in an XML response with the following tags:

Output	XML Tag	Comments
Type of Response	<AddressValidateResponse>	-
Address Verification Number	<Address ID='#'>	-
Name of Firm	<FirmName>	If provided in request.
Address Line 1	<Address1>	-
Address Line 2	<Address2>	-
City	<City>	If the city name is greater than 14 characters, the city abbreviation is returned.
State	<State>	-
ZIP Code	<Zip5>	-
ZIP Code + 4	<Zip4>	-
Message when multiple addresses found	<ReturnText>	This output is only returned when the address entered results in multiple locations being found by the Shipping API server, but a default address exists. The text of the message will read: “Default address: The address you entered was found but more information is needed (such as an apartment, suite, or box number) to match to a specific address.”

“Canned” Test Responses

For your test to be successful, the following responses should be returned *verbatim*. If any values were changes in your request, the following default error will occur:

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="0">
    <Error>
      <Number>-2147219040</Number>
      <Source>SQLServerTest;SQLServerTest.CallAddressDll</Source>
      <Description>This Information has not been included in this Test
      Server.</Description>
      <HelpFile></HelpFile>
      <HelpContext></HelpContext>
    </Error>
  </Address>
</AddressValidateResponse>
```

Although the input may be valid, the response will still raise this error, because those particular values have not been included in this test server. Refer to the *Errors* section for an explanation of any other returned errors.

Response to Valid Test Request #1

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="0">
    <Address2>6406 IVY LN</Address2>
    <City>GREENBELT</City>
    <State>MD</State>
    <Zip5>20770</Zip5>
    <Zip4>1440</Zip4>
  </Address>
</AddressValidateResponse>
```

Response to Valid Test Request #2

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="1">
    <Address2>8 WILDWOOD DR</Address2>
    <City>OLD LYME</City>
    <State>CT</State>
    <Zip5>06371</Zip5>
    <Zip4>1844</Zip4>
  </Address>
</AddressValidateResponse>
```

Response to Valid Test Request #3

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="2">
    <Address2>4411 ROMLON ST</Address2>
    <City>BELTSVILLE</City>
    <State>MD</State>
    <Zip5>20705</Zip5>
    <Zip4>2425</Zip4>
  </Address>
</AddressValidateResponse>
```

Response to Valid Test Request #4

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="3">
    <Address2>3527 SHARONWOOD RD APT 3C</Address2>
    <City>LAUREL</City>
    <State>MD</State>
    <Zip5>20724</Zip5>
    <Zip4>5920</Zip4>
  </Address>
</AddressValidateResponse>
```

Response to Pre-defined Error Request #1: *"The Address Could Not Be Found"*

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="0">
    <Error>
      <Number>-2147219401</Number>
      <Source>SOLServerTest;SOLServerTest.CallAddressDll</Source>
      <Description>That address could not be found.</Description>
      <HelpFile></HelpFile>
      <HelpContext></HelpContext>
    </Error>
  </Address>
</AddressValidateResponse>
```

Response to Pre-defined Error Request #2: *"Multiple Addresses Were Found and There Is No Default Available"*

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="0">
    <Error>
      <Number>-2147219403</Number>
      <Source>SOLServerTest;SOLServerTest.CallAddressDll</Source>
      <Description>Multiple responses found. No default
address.</Description>
      <HelpFile></HelpFile>
      <HelpContext></HelpContext>
    </Error>
  </Address>
</AddressValidateResponse>
```

Response to Pre-defined Error Request #3: *"The State is Invalid"*

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="0">
    <Error>
      <Number>-2147219402</Number>
      <Source>SOLServerTest;SOLServerTest.CallAddressDll</Source>
      <Description>That State is not valid.</Description>
      <HelpFile></HelpFile>
      <HelpContext></HelpContext>
    </Error>
  </Address>
</AddressValidateResponse>
```

Response to Pre-defined Error Request #4: *"The City is Invalid"*

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="0">
    <Error>
      <Number>-2147219400</Number>
      <Source>SOLServerTest;SOLServerTest.CallAddressDll</Source>
      <Description>That is not a valid city.</Description>
      <HelpFile></HelpFile>
      <HelpContext></HelpContext>
    </Error>
  </Address>
</AddressValidateResponse>
```

"Live" Responses

XML Output Example

```
<AddressValidateResponse>
  <Address ID="0">
    <FirmName>XYZ Corp.</FirmName>
    <Address2>6406 IVY LN</Address2>
    <City>GREENBELT</City>
    <State>MD</State>
    <Zip5>20770</Zip5>
    <Zip4>1440</Zip4>
  </Address>
</AddressValidateResponse>
```

Transaction Procedures for ZIP Code Lookup API

The illustration below shows the transactional flow of information to and from the USPS ZIP Code Lookup API server.

ZIP Code Lookup API Server



Technical Steps

Step 1: Build the XML Request

“Canned” Test Requests

For testing purposes, the only values in the test code in this section that you should change are the “userid” and “password.” Enter the user ID and password you received in the registration e-mail. *All remaining code in the test scripts provided below must remain unchanged.*

All of the test script code contained in this document can be cut and pasted for your use in testing the software. To copy the test script code from this PDF file, click on the icon for “Text Selector” and highlight the code. (The icon will look like



or



depending on your version of Adobe Acrobat.) You can then copy the code and paste it into your test document.

Valid Test Requests

There are four valid requests included in this procedure:

Valid Test Request #1

```
http://SERVERNAME/ShippingAPITest.dll?API=ZipCodeLookup&XML=<ZipCodeLookupRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="0"><Address1></Address1><Address2>6406 Ivy Lane</Address2><City>Greenbelt</City><State>MD</State></Address></ZipCodeLookupRequest>
```

Valid Test Request #2

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<ZipCodeLookupRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="1"><Address1></Address1><Address2>8 Wildwood Drive</Address2><City>Old Lyme</City><State>CT</State><Zip5>06371</Zip5><Zip4></Zip4></Address></ZipCodeLookupRequest>
```

Valid Test Request #3

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<ZipCodeLookupRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="2"><Address1></Address1><Address2>4411 Romlon Street</Address2><City>Beltsville</City><State>MD</State><Zip5></Zip5><Zip4></Zip4></Address></ZipCodeLookupRequest>
```

Valid Test Request #4

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<ZipCodeLookupRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="3"><Address1></Address1><Address2>3527 Sharonwood Road Apt. 3C</Address2><City>Laurel</City><State>MD</State><Zip5></Zip5><Zip4></Zip4></Address></ZipCodeLookupRequest>
```

Pre-Defined Error Requests

There are four pre-defined errors included for this procedure. Be sure to note the request numbers so you can match up the responses you will receive as provided in the “Canned” Test Responses section.

Pre-defined Error Request #1: “The Address Could Not Be Found”

For testing purposes, this error will occur when the state input is “DE.”

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<ZipCodeLookupRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="0"><Address1></Address1><Address2>3527 Sharonwood Road Apt. 3C</Address2><City>Wilmington</City><State>DE</State><Zip5></Zip5><Zip4></Zip4></Address></ZipCodeLookupRequest>
```

Pre-defined Error Request #2:

“Multiple Addresses Were Found and There Is No Default Available”

For testing purposes, this error will occur when the state input is “DC.”

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<ZipCodeLookupRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="0"><Address1></Address1><Address2>1600 Pennsylvania Avenue</Address2><City>Washington</City><State>DC</State><Zip5></Zip5><Zip4></Zip4></Address></ZipCodeLookupRequest>
```

Pre-defined Error Request #3: “The State is Invalid”

This error will occur when the state input is “ZZ.”

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<ZipCodeLookupRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="0"><Address1></Address1><Address2>123 Main Street</Address2><City>Washington</City><State>ZZ</State><Zip5></Zip5><Zip4></Zip4></Address></ZipCodeLookupRequest>
```

Pre-defined Error Request #4: “The City Is Invalid”

This error will occur when the state input is "NJ."

```
http://SERVERNAME/ShippingAPITest.dll?API=Verify&XML=<ZipCodeLookupRequest%20
USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><Address ID="0"><Address1></Address1>
<Address2>123 Main Street</Address2><City>Trenton</City><State>NJ</State>
<Zip5></Zip5><Zip4></Zip4></Address></ZipCodeLookupRequest>
```

“Live” Request

Refer to the “Canned” Test Requests section above for instructions on how to cut and paste the sample code from this PDF file.

Remember that you are provided with a different server name to send “live” requests.

When building the XML request, pay particular attention to the *order and case* for tags.

The table below presents the *required* XML input tags for generating “Live” requests and the restrictions on the values allowed. An error message will be returned if the tag does not contain a value or if an incorrect value is entered. Also, be aware of the maximum character amounts allowed for some tags. If the user enters more than those amounts, an error will not be generated. **The API will simply pass in the characters up to the maximum amount allowed and disregard the rest.** This is important since the resulting value could prevent delivery.



Developers: For sample code utilizing Perl and ASP, refer to the Domestic Rates Calculator API and Track/Confirm API user’s guides.

Input	XML Tag	Values Allowed
Type of Request	<ZipCodeLookupRequest...	Input tag exactly as presented.
User ID	...USERID="userid"...	Use user ID provided with registration.
Password	...PASSWORD="password">	Use password provided with registration.
Address Lookup Number	<Address ID=' #'>	Up to five address verifications can be included per transaction.
Name of Firm	<FirmName>	Providing the firm name tag is optional. Maximum characters allowed: 38.
Address Line 1	<Address1>	Address Line 1 is used to provide an apartment or suite number, if applicable. If not applicable, include the open and close tag with no input. Maximum characters allowed: 38.
Address Line 2	<Address2>	This tag is required for this API. Maximum characters allowed: 38.
City	<City>	Maximum characters allowed: 15. This tag is required for this API.
State	<State>	Maximum characters allowed: 2. This tag is required for this API.

The “Live” XML request should be in the form:

```
<ZipCodeLookupRequest USERID="xxxxxxx" PASSWORD="xxxxxxx">
  <Address ID='0'>
    <FirmName></FirmName>
    <Address1>Suite #</Address1>
    <Address2>Street Address</Address2>
    <City></City>
    <State></State>
  </Address>
  <Address ID='1'>
    <FirmName></FirmName>
    <Address1>Apt/Suite No</Address1>
    <Address2>Street Address</Address2>
    <City></City>
    <State></State>
  </Address>
</ZipCodeLookupRequest>
```

Visual Basic Request

Using the Microsoft XML object model in Visual Basic, such a request can be built as shown below. In this code sample, the data needed to build the XML is obtained from a form. The <ServiceType> element is obtained from an option button control and the <ImageType> is from a combo box control. All other fields are obtained from text box controls.

```
Dim oXMLDocument As DOMDocument
Dim oRequestLevel As IXMLDOMElement
Dim oAddressLevel As IXMLDOMElement
Dim oAddressElementLevel As IXMLDOMElement

' Build the XML Request

' Create a new XML document
Set oXMLDocument = New DOMDocument

' Build the top-level (request)
Set oRequestLevel =
oXMLDocument.createElement("ZipCodeLookupRequest")
oRequestLevel.setAttribute "USERID", "MyUserId"
oRequestLevel.setAttribute "PASSWORD", "MyPassword"

' Add one or more Address levels
Set oAddressLevel = oXMLDocument.createElement("Address")
oAddressLevel.setAttribute "ID", "0"
oRequestLevel.appendChild oAddressLevel

' Address1
Set oAddressElementLevel =
oXMLDocument.createElement("Address1")<BR>
oAddressElementLevel.appendChild
oXMLDocument.createTextNode("")
oAddressLevel.appendChild oAddressElementLevel

' Address2
Set oAddressElementLevel =
oXMLDocument.createElement("Address2")
oAddressElementLevel.appendChild
oXMLDocument.createTextNode("123 Main Street ")
```

```
oAddressLevel.appendChild oAddressElementLevel

' City
Set oAddressElementLevel =
oXMLDocument.createElement("City")
oAddressElementLevel.appendChild
oXMLDocument.createTextNode("Somewhere")
oAddressLevel.appendChild oAddressElementLevel

' State
Set oAddressElementLevel =
oXMLDocument.createElement("State")
oAddressElementLevel.appendChild
oXMLDocument.createTextNode("MD")
oAddressLevel.appendChild oAddressElementLevel

' Append address level to request level
oRequestLevel.appendChild oAddressLevel

' Append request level to document
oXMLDocument.appendChild oRequestLevel
```

Steps 2 & 3: Make the Internet Connection and Send the XML Request

These two steps are presented together to simplify things. The two steps actually involve four separate functions:

1. Making the connection to the USPS Shipping API server (test or production server)
2. Sending the request (whether Visual Basic, Perl, ASP, or any other language)
3. Receiving the response from the API server
4. Closing the Internet connection

These steps are identical for sending “Canned” test requests or “Live” requests. **Remember, however, that you are provided with a different server name to send “live” requests.**

This section provides two samples to make the Internet connection. This is not an all-inclusive list. It simply represents the most common and easiest ways to make the Internet connection.

- Using the USPS-supplied HTTP Connection DLL

The HTTP Connection DLL is recommended for NT systems. This software, created specifically for the USPS API implementation, provides e-tailers with a thread-safe sockets interface to submit XML requests and receive XML responses from the API server.

- Using Microsoft's WinInet

Although you can use the WinInet DLL to make the connection to the API server, it is not recommended for server applications due to limitations in the DLL. It is recommended that you either use the USPS-supplied HTTP Connection DLL or write your own sockets interface that can be used to make multiple connections and will remain thread-safe.

Using HTTP Connection DLL

To obtain this code you must submit a Licensing Agreement. See the *Administrative Guide for APIs* for the agreement.

Using WinInet

This sample code shows how to use Microsoft's WinInet dll to make the Internet connection, using either the "GET" or "POST" (necessary for requests over 2K in size) methods. XMLSTRING represents the URL-encoded XML request and SERVERNAME indicates the name of the USPS web site to which you are connecting. For more information on the Microsoft WinInet product, go to http://msdn.microsoft.com/library/techart/msdn_vbhttp.htm.

Although you can use the WinInet dll to make the connection to the API server, *it is not recommended for server applications due to limitations in the dll*. It is recommended that you write a sockets interface that can be used to make multiple connections and will remain thread-safe.

```
Dim hOpen As Long, hConnection As Long, hFile As Long, numread As Long
Dim File As String, xml As String, sHeader As String, htmlFile As String, tmp
As String * 2048
Dim bDoLoop As Boolean

File = "/ShippingAPI.dll?"
xml = "API=ZipCodeLookup&XML=" & XMLSTRING

hOpen = InternetOpen("", 1, vbNullString, vbNullString, 0)

hConnection = InternetConnect(hOpen, SERVERNAME, 0, _
    "", "", 3, 0, 0)

'.....
'get
'File = File & xml
'hFile = HttpOpenRequest(hConnection, "GET", File, "HTTP/1.0", vbNullString,
0, 0, 0)
'OR
'.....

'.....
' post
'hFile = HttpOpenRequest(hConnection, "POST", File, "HTTP/1.0", vbNullString,
0, 0, 0)

sHeader = "Content-Type: application/x-www-form-urlencoded" _
    & vbCrLf

Call HttpAddRequestHeaders(hFile, _
    sHeader, Len(sHeader), 0)
'.....

bDoLoop = HttpSendRequest(hFile, vbNullString, 0, xml, Len(xml))

bDoLoop = True
While bDoLoop
```

```
tmp = vbNullString
bDoLoop = InternetReadFile(hFile, tmp, Len(tmp), numread)
If Not bDoLoop Then
    Exit Sub
Else
    htmlFile = htmlFile & Left$(tmp, numread)
    If Not CBool(numread) Then bDoLoop = False
End If
Wend

If hFile <> 0 Then InternetCloseHandle (hFile)
If hConnection <> 0 Then InternetCloseHandle (hConnection)
If hOpen <> 0 Then InternetCloseHandle (hOpen)
```

Step 4: Unpack the XML Response

This step is identical for unpacking “Canned” test responses or “Live” responses.

Types of Responses

When the USPS Shipping API returns a response, it will either return a successful response document or an error document. Anytime you receive a response, you should check to see if the document is <Error>. Refer to the *Errors* section.

Using Visual Basic

Using the Microsoft XML object model in Visual Basic, such responses can be unpacked as follows:

```
Const sXML_RESPONSE As String = "<?xml version='1.0'?>" & _
    "<ZipCodeLookupResponse>" & _
    "<Address ID='0'>" & _
    "<Address2></Address2>" & _
    "<City></City>" & _
    "<State></State>" & _
    "<Zip5></Zip5>" & _
    "<Zip4></Zip4>" & _
    "</Address>" & _
    "</ZipCodeLookupResponse>"

Dim xmlDoc As DOMDocument
Dim nodeList As IXMLDOMNodeList
Dim n As IXMLDOMNode, e As IXMLDOMNode, t As IXMLDOMNode
Dim i As Integer, j As Integer, k As Integer

Dim sZip5 As String
Dim sZip4 As String

Dim lErrorNumber As Long
Dim sDescription As String
Dim sSource As String
Dim sHelpFile As String
Dim sHelpContextId As String
```

```
Set xmlDoc = New DOMDocument
xmlDoc.validateOnParse = False
xmlDoc.loadXML (sXML_RESPONSE) 'Response
If xmlDoc.documentElement.nodeName = "Error" Then 'Top-level
Error
    Set nodeList = xmlDoc.getElementsByTagName("Error")
    Call UnpackErrorNode(nodeList.Item(0), lErrorNumber,
    sDescription, sSource, sHelpFile, sHelpContextId)
    ' Add code here to display the error
Else 'no Top-level Error
Set nodeList = xmlDoc.getElementsByTagName("Address")
For i = 0 To nodeList.length - 1
    Set n = nodeList.Item(i)
    For j = 0 To n.childNodes.length - 1
        Set e = n.childNodes.Item(j)
        If e.nodeName = "Error" Then 'Lower-level error
            Call UnpackErrorNode(e, lErrorNumber,
            sDescription, sSource, sHelpFile, sHelpContextId)
            ' Add code here to display the error
        Else 'No error in Package
            Select Case e.nodeName
                Case "Zip5"
                    If e.hasChildNodes Then
                        sZip5 = e.firstChild.nodeValue
                    End If
                Case "Zip4"
                    If e.hasChildNodes Then
                        sZip4 = e.firstChild.nodeValue
                    End If
            End Select
        End If
    Next j
Next i
End If
```

The UnpackErrorNode common subroutine that is referred to in the above code examples unpacks an XML Error node into individual variables.

```
' Input:
'   oNode - XML Error Node
' Output:
'   lErrorNumber - Error Number
'   sDescription - Error Description
'   sSource - Error Source
'   sHelpFile - Help File Name
'   sHelpContextId - Help Context Id
```

```
Private Sub UnpackErrorNode(ByRef oNode As IXMLDOMNode, ByRef lErrorNumber As
Long, ByRef sDescription As String, ByRef sSource As String, ByRef sHelpFile
As String, ByRef sHelpContextId As String)
```

```
    Dim oNodeError As IXMLDOMNode
```

```
    Dim lIndex As Long
```

```
    lErrorNumber = 0
```

```
    sSource = ""
```

```
sDescription = ""
sHelpFile = ""
sHelpContextId = ""

For lIndex = 0 To oNode.childNodes.length - 1
  Set oNodeError = oNode.childNodes.Item(lIndex)
  Select Case oNodeError.nodeName
    Case "Source"
      If oNodeError.hasChildNodes Then
        sSource = oNodeError.firstChild.nodeValue
      End If
    Case "Number"
      If oNodeError.hasChildNodes Then
        lErrorNumber = oNodeError.firstChild.nodeValue
      End If
    Case "Description"
      If oNodeError.hasChildNodes Then
        sDescription = oNodeError.firstChild.nodeValue
      End If
    Case "HelpFile"
      If oNodeError.hasChildNodes Then
        sHelpFile = oNodeError.firstChild.nodeValue
      End If
    Case "HelpContext"
      If oNodeError.hasChildNodes Then
        sHelpContextId =
          oNodeError.firstChild.nodeValue
      End If
  End Select
Next
End Sub
```

Errors

Error conditions are handled at the main XML document level. For APIs that can handle multiple transactions, the error conditions for requests for multiple responses to be returned together are handled at the response level. For example: an API developer sends a request for rates for two packages. If the addresses are non-existent, an “Error document” is returned to the user. On the other hand, if the address for the first package is acceptable but not the second, the response document contains the information for the first address, but under the XML tag for the second address there is an error tag.

Error documents follow the Visual Basic error standards and have following format:

```
<ZipCodeLookupResponse>
  <Address ID="0">
    <Error>
      <Number></Number>
      <Source></Source>
      <Description></Description>
      <HelpFile></HelpFile>
      <HelpContext></HelpContext>
    </Address>
  </ZipCodeLookupResponse>
```

where:

- Number = the error number generated by the API server
- Source = the component and interface that generated the error on the API server
- Description = the error description
- HelpFile = [reserved]
- HelpContext = [reserved]

Errors that are further down in the hierarchy also follow the above format.

Output

After following Technical Step 4 and unpacking the XML response, you will have the output from your request. This section describes the different outputs resulting from “Canned” test requests and “Live” requests. Both types of requests result in an XML response with the following tags:

Output	XML Tag	Comments
Type of Response	<ZipCodeLookupResponse>	-
Address ID Number	<Address ID='# '>	-
Name of Firm	<FirmName>	Will be returned only if tag was included in the XML request.
Address Line 1	<Address1>	-
Address Line 2	<Address2>	-
City	<City>	If the city name is greater than 14 characters, the city abbreviation is returned.
State	<State>	-
ZIP Code	<Zip5>	-
ZIP Code + 4	<Zip4>	-

“Canned” Test Responses

For your test to be successful, the following responses should be returned *verbatim*. If any values were changes in your request, the following default error will occur:

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="0">
    <Error>
      <Number>-2147219040</Number>
      <Source>SOLServerTest;SOLServerTest.CallAddressDll</Source>
      <Description>This Information has not been included in this Test
      Server.</Description>
      <HelpFile></HelpFile>
      <HelpContext></HelpContext>
    </Error>
  </Address>
</AddressValidateResponse>
```

Although the input may be valid, the response will still raise this error, because those particular values have not been included in this test server. Refer to the *Errors* section for an explanation of any other returned errors.

Response to Valid Test Request #1

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="0">
    <Address2>6406 IVY LN</Address2>
    <City>GREENBELT</City>
    <State>MD</State>
    <Zip5>20770</Zip5>
    <Zip4>1440</Zip4>
  </Address>
</AddressValidateResponse>
```

Response to Valid Test Request #2

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="1">
    <Address2>8 WILDWOOD DR</Address2>
    <City>OLD LYME</City>
    <State>CT</State>
    <Zip5>06371</Zip5>
    <Zip4>1844</Zip4>
  </Address>
</AddressValidateResponse>
```

Response to Valid Test Request #3

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="2">
    <Address2>4411 ROMLON ST</Address2>
    <City>BELTSVILLE</City>
    <State>MD</State>
    <Zip5>20705</Zip5>
    <Zip4>2425</Zip4>
  </Address>
</AddressValidateResponse>
```

Response to Valid Test Request #4

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="3">
    <Address2>3527 SHARONWOOD RD APT 3C</Address2>
    <City>LAUREL</City>
    <State>MD</State>
    <Zip5>20724</Zip5>
    <Zip4>5920</Zip4>
  </Address>
</AddressValidateResponse>
```

Response to Pre-defined Error Request #1: "The Address Could Not Be Found"

```
<?xml version="1.0"?>
<AddressValidateResponse>
  <Address ID="0">
    <Error>
      <Number>-2147219401</Number>
      <Source>SQLServerTest;SQLServerTest.CallAddressDll</Source>
    </Error>
  </Address>
</AddressValidateResponse>
```

```
        <Description>That address could not be found.</Description>
        <HelpFile></HelpFile>
        <HelpContext></HelpContext>
    </Error>
</Address>
</AddressValidateResponse>
```

**Response to Pre-defined Error Request #2:
“Multiple Addresses Were Found and There Is No Default Available”**

```
<?xml version="1.0"?>
<AddressValidateResponse>
    <Address ID="0">
        <Error>
            <Number>-2147219403</Number>
            <Source>SQLServerTest;SQLServerTest.CallAddressDll</Source>
            <Description>Multiple responses found. No default
            address.</Description>
            <HelpFile></HelpFile>
            <HelpContext></HelpContext>
        </Error>
    </Address>
</AddressValidateResponse>
```

Response to Pre-defined Error Request #3: “The State is Invalid”

```
<?xml version="1.0"?>
<AddressValidateResponse>
    <Address ID="0">
        <Error>
            <Number>-2147219402</Number>
            <Source>SQLServerTest;SQLServerTest.CallAddressDll</Source>
            <Description>That State is not valid.</Description>
            <HelpFile></HelpFile>
            <HelpContext></HelpContext>
        </Error>
    </Address>
</AddressValidateResponse>
```

Response to Pre-defined Error Request #4: “The City is Invalid”

```
<?xml version="1.0"?>
<AddressValidateResponse>
    <Address ID="0">
        <Error>
            <Number>-2147219400</Number>
            <Source>SQLServerTest;SQLServerTest.CallAddressDll</Source>
            <Description>That is not a valid city.</Description>
            <HelpFile></HelpFile>
            <HelpContext></HelpContext>
        </Error>
    </Address>
</AddressValidateResponse>
```

“Live” Responses

XML Output Example

```
<ZipCodeLookupResponse>
```

```
<Address ID="0">
  <FirmName></FirmName>
  <Address1></Address1>
  <Address2></Address2>
  <City></City>
  <State></State>
  <Zip5></Zip5>
  <Zip4></Zip4>
</Address>
<Address ID="1">
  <FirmName></FirmName>
  <Address1></Address1>
  <Address2></Address2>
  <City></City>
  <State></State>
  <Zip5></Zip5>
  <Zip4></Zip4>
</Address>
</ZipCodeLookupResponse>
```

Transaction Procedures for City/State Lookup API

The illustration below shows the transactional flow of information to and from the USPS City/State Lookup API server.

City/State Lookup API Server



Technical Steps

Step 1: Build the XML Request

“Canned” Test Requests

For testing purposes, the only values in the test code in this section that you should change are the “userid” and “password.” Enter the user ID and password you received in the registration e-mail. *All remaining code in the test scripts provided below must remain unchanged.*

All of the test script code contained in this document can be cut and pasted for your use in testing the software. To copy the test script code from this PDF file, click on the icon for “Text Selector” and highlight the code. (The icon will look like



or



depending on your version of Adobe Acrobat.) You can then copy the code and paste it into your test document.

Valid Test Requests

There are five valid requests included in this procedure:

Valid Test Request #1

```
http://SERVERNAME/ShippingAPITest.dll?API=CityStateLookup&XML=<CityStateLooku
pRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><ZipCode ID= "0">
<Zip5>90210</Zip5></ZipCode></CityStateLookupRequest>
```

Valid Test Request #2

```
http://SERVERNAME/ShippingAPITest.dll?API=CityStateLookup&XML=<CityStateLooku
pRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><ZipCode ID= "0">
<Zip5>20770</Zip5></ZipCode></CityStateLookupRequest>
```

Valid Test Request #3

```
http://SERVERNAME/ShippingAPITest.dll?API=CityStateLookup&XML=<CityStateLooku
pRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><ZipCode ID= "0">
<Zip5>21113</Zip5></ZipCode></CityStateLookupRequest>
```

Valid Test Request #4

```
http://SERVERNAME/ShippingAPITest.dll?API=CityStateLookup&XML=<CityStateLooku
pRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><ZipCode ID= "0">
<Zip5>21032</Zip5></ZipCode></CityStateLookupRequest>
```

Valid Test Request #5

```
http://SERVERNAME/ShippingAPITest.dll?API=CityStateLookup&XML=<CityStateLooku
pRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><ZipCode ID= "0">
<Zip5>21117</Zip5></ZipCode></CityStateLookupRequest>
```

Pre-Defined Error Request

Invalid ZIP Code

This error will occur if the ZIP Code input is "99999."

```
http://SERVERNAME/ShippingAPITest.dll?API=CityStateLookup&XML=<CityStateLooku
pRequest%20USERID="xxxxxxx"%20PASSWORD="xxxxxxx"><ZipCode ID= "0">
<Zip5>99999</Zip5></ZipCode></CityStateLookupRequest>
```

"Live" Request

Refer to the "Canned" Test Requests section above for instructions on how to cut and paste the sample code from this PDF file.

Remember that you are provided with a different server name to send "live" requests.

When building the XML request, pay particular attention to the *order and case* for tags.

The table below presents the *required* XML input tags for generating "Live" requests and the restrictions on the values allowed. An error message will be returned if the tag does not contain a value or if an incorrect value is entered. Also, be aware of the maximum character amounts allowed for some tags. If the user enters more than those amounts, an error will not be generated. **The API will simply pass in the characters up to the maximum amount allowed and disregard the rest.** This is important since the resulting value could prevent delivery.



Developers: For sample code utilizing Perl and ASP, refer to the Domestic Rates Calculator API and Track/Confirm API user's guides.

Input	XML Tag	Values Allowed
Type of Request	<CityStateLookupRequest...	Input tag exactly as presented.
User ID	...USERID="userid"...	Use user ID provided with registration.
Password	...PASSWORD="password">	Use password provided with registration.
ZIP Code Lookup Number	<ZipCode ID='#'>	Up to five ZIP Codes can be included per transaction.
ZIP Code of City or State	<Zip5>	Input tag exactly as presented, not all caps. Maximum characters allowed: 5

The “Live” XML request should be in the form:

```
<CityStateLookupRequest USERID="xxxxxxxx" PASSWORD="xxxxxxxx">
  <ZipCode ID="0">
    <Zip5>90210</Zip5>
  </ZipCode>
  <ZipCode ID="1">
    <Zip5>20770</Zip5>
  </ZipCode>
</CityStateLookupRequest>
```

Visual Basic Request

Using the Microsoft XML object model in Visual Basic, such a request can be built as shown below. In this code sample, the data needed to build the XML is obtained from a form. The <ServiceType> element is obtained from an option button control and the <ImageType> is from a combo box control. All other fields are obtained from text box controls.

```
Dim oXMLDocument As DOMDocument
Dim oRequestLevel As IXMLDOMElement
Dim oLookupLevel As IXMLDOMElement
Dim oLookupElementLevel As IXMLDOMElement

' Build the XML Request

' Create a new XML document
Set oXMLDocument = New DOMDocument

' Build the top-level (request)
Set oRequestLevel =
oXMLDocument.createElement("ExpressMailRequest")
oRequestLevel.setAttribute "USERID", "MyUserId"
oRequestLevel.setAttribute "PASSWORD", "MyPassword"

' Add one or more Address levels
Set oLookupLevel = oXMLDocument.createElement("ZipCode")
oLookupLevel.setAttribute "ID", "0"
oRequestLevel.appendChild oLookupLevel

' Zip Code
Set oLookupElementLevel =
oXMLDocument.createElement("Zip5")
oLookupElementLevel.appendChild
oXMLDocument.createTextNode("90210")
oLookupLevel.appendChild oLookupElementLevel

' Append lookup level to request
```

```
oRequestLevel.appendChild oLookupLevel  
  
' Append request level to document  
oXMLDocument.appendChild oRequestLevel
```

Steps 2 & 3: Make the Internet Connection and Send the XML Request

These two steps are presented together to simplify things. The two steps actually involve four separate functions:

1. Making the connection to the USPS Shipping API server (test or production server)
2. Sending the request (whether Visual Basic, Perl, ASP, or any other language)
3. Receiving the response from the API server
4. Closing the Internet connection

These steps are identical for sending “Canned” test requests or “Live” requests. **Remember, however, that you are provided with a different server name to send “live” requests.**

This section provides two samples to make the Internet connection. This is not an all-inclusive list. It simply represents the most common and easiest ways to make the Internet connection.

- Using the USPS-supplied HTTP Connection DLL

The HTTP Connection DLL is recommended for NT systems. This software, created specifically for the USPS API implementation, provides e-tailers with a thread-safe sockets interface to submit XML requests and receive XML responses from the API server.

- Using Microsoft's WinInet

Although you can use the WinInet DLL to make the connection to the API server, it is not recommended for server applications due to limitations in the DLL. It is recommended that you either use the USPS-supplied HTTP Connection DLL or write your own sockets interface that can be used to make multiple connections and will remain thread-safe.

Using HTTP Connection DLL

To obtain this code you must submit a Licensing Agreement. See the *Administrative Guide for APIs* for the agreement.

Using WinInet

This sample code shows how to use Microsoft's WinInet dll to make the Internet connection, using either the “GET” or “POST” (necessary for requests over 2K in size) methods. XMLSTRING represents the URL-encoded XML request and SERVERNAME indicates the name of the USPS web site to which you are connecting. For more information on the Microsoft WinInet product, go to http://msdn.microsoft.com/library/techart/msdn_vbhttp.htm.

Although you can use the WinInet dll to make the connection to the API server, *it is not recommended for server applications due to limitations in the dll.* It is recommended that you write a sockets interface that can be used to make multiple connections and will remain thread-safe.

```
Dim hOpen As Long, hConnection As Long, hFile As Long, numread As Long
```

```
Dim File As String, xml As String, sHeader As String, htmlFile As String, tmp
As String * 2048
Dim bDoLoop As Boolean

File = "/ShippingAPI.dll?"
xml = "API=CityStateLookup&XML=" & XMLSTRING

hOpen = InternetOpen("", 1, vbNullString, vbNullString, 0)

hConnection = InternetConnect(hOpen, SERVERNAME, 0, _
    "", "", 3, 0, 0)

'.....
'get
'File = File & xml
'hFile = HttpOpenRequest(hConnection, "GET", File, "HTTP/1.0", vbNullString,
0, 0, 0)
'OR
'.....

'.....
' post
'hFile = HttpOpenRequest(hConnection, "POST", File, "HTTP/1.0", vbNullString,
0, 0, 0)

sHeader = "Content-Type: application/x-www-form-urlencoded" _
    & vbCrLf

Call HttpAddRequestHeaders(hFile, _
    sHeader, Len(sHeader), 0)
'.....

bDoLoop = HttpSendRequest(hFile, vbNullString, 0, xml, Len(xml))

bDoLoop = True
While bDoLoop
    tmp = vbNullString
    bDoLoop = InternetReadFile(hFile, tmp, Len(tmp), numread)
    If Not bDoLoop Then
        Exit Sub
    Else
        htmlFile = htmlFile & Left$(tmp, numread)
        If Not CBool(numread) Then bDoLoop = False
    End If
Wend

If hFile <> 0 Then InternetCloseHandle (hFile)
If hConnection <> 0 Then InternetCloseHandle (hConnection)
If hOpen <> 0 Then InternetCloseHandle (hOpen)
```

Step 4: Unpack the XML Response

This step is identical for unpacking “Canned” test responses or “Live” responses.

Types of Responses

When the USPS Shipping API returns a response, it will either return a successful response document or an error document. Anytime you receive a response, you should check to see if the document is <Error>. Refer to the *Errors* section.

Using Visual Basic

Using the Microsoft XML object model in Visual Basic, such responses can be unpacked as follows:

```
Const sXML_RESPONSE As String = "<?xml version='1.0'?>" & _
    "<CityStateLookupResponse>" & _
    "<ZipCode ID='0'" & _
    "<Zip5></Zip5>" & _
    "<City></City>" & _
    "<State></State>" & _
    "</ZipCode>" & _
    "</CityStateLookupResponse>"

Dim xmlDoc As DOMDocument
Dim nodeList As IXMLDOMNodeList
Dim n As IXMLDOMNode, e As IXMLDOMNode, t As IXMLDOMNode
Dim i As Integer, j As Integer, k As Integer

Dim sCity As String
Dim sState As String

Dim lErrorNumber As Long
Dim sDescription As String
Dim sSource As String
Dim sHelpFile As String
Dim sHelpContextId As String

Set xmlDoc = New DOMDocument
xmlDoc.validateOnParse = False
xmlDoc.loadXML (sXML_RESPONSE) 'Response
If xmlDoc.documentElement.nodeName = "Error" Then 'Top-level
Error
    Set nodeList = xmlDoc.getElementsByTagName("Error")
    Call UnpackErrorNode (nodeList.Item(0), lErrorNumber,
sDescription, sSource, sHelpFile, sHelpContextId)
    ' Add code here to display the error
Else 'no Top-level Error
    Set nodeList = xmlDoc.getElementsByTagName("ZipCode")
    For i = 0 To nodeList.length - 1
        Set n = nodeList.Item(i)
        For j = 0 To n.childNodes.length - 1
            Set e = n.childNodes.Item(j)
            If e.nodeName = "Error" Then 'Lower-level error
                Call UnpackErrorNode(e, lErrorNumber,
sDescription, sSource, sHelpFile, sHelpContextId)
                ' Add code here to display the error
            Else 'No error in Package
                Select Case e.nodeName
```

```
        Case "City"
            If e.hasChildNodes Then
                sCity = e.firstChild.nodeValue
            End If
        Case "State"
            If e.hasChildNodes Then
                sState = e.firstChild.nodeValue
            End If
        End Select
    End If
Next j
Next i
End If
```

The UnpackErrorNode common subroutine that is referred to in the above code examples unpacks an XML Error node into individual variables.

```
' Input:
'   oNode - XML Error Node
' Output:
'   lErrorNumber - Error Number
'   sDescription - Error Description
'   sSource - Error Source
'   sHelpFile - Help File Name
'   sHelpContextId - Help Context Id
```

```
Private Sub UnpackErrorNode(ByRef oNode As IXMLDOMNode, ByRef lErrorNumber As Long, ByRef sDescription As String, ByRef sSource As String, ByRef sHelpFile As String, ByRef sHelpContextId As String)
```

```
    Dim oNodeError As IXMLDOMNode

    Dim lIndex As Long

    lErrorNumber = 0
    sSource = ""
    sDescription = ""
    sHelpFile = ""
    sHelpContextId = ""

    For lIndex = 0 To oNode.childNodes.length - 1
        Set oNodeError = oNode.childNodes.Item(lIndex)
        Select Case oNodeError.nodeName
            Case "Source"
                If oNodeError.hasChildNodes Then
                    sSource = oNodeError.firstChild.nodeValue
                End If
            Case "Number"
                If oNodeError.hasChildNodes Then
                    lErrorNumber = oNodeError.firstChild.nodeValue
                End If
            Case "Description"
                If oNodeError.hasChildNodes Then
                    sDescription = oNodeError.firstChild.nodeValue
                End If
            Case "HelpFile"
                If oNodeError.hasChildNodes Then
```

```
        sHelpFile = oNodeError.firstChild.nodeValue
    End If
    Case "HelpContext"
        If oNodeError.hasChildNodes Then
            sHelpContextId =
                oNodeError.firstChild.nodeValue
        End If
    End Select
Next
End Sub
```

Errors

Error conditions are handled at the main XML document level. For APIs that can handle multiple transactions, the error conditions for requests for multiple responses to be returned together are handled at the response level. For example: an API developer sends a request for rates for two packages. If the addresses are non-existent, an “Error document” is returned to the user. On the other hand, if the address for the first package is acceptable but not the second, the response document contains the information for the first address, but under the XML tag for the second address there is an error tag.

Error documents follow the Visual Basic error standards and have following format:

```
<CityStateLookupResponse>
  <ZipCode ID="0">
    <Error>
      <Number></Number>
      <Source></Source>
      <Description></Description>
      <HelpFile></HelpFile>
      <HelpContext></HelpContext>
    </ZipCode>
  </CityStateLookupResponse>
```

where:

- Number = the error number generated by the API server
- Source = the component and interface that generated the error on the API server
- Description = the error description
- HelpFile = [reserved]
- HelpContext = [reserved]

Errors that are further down in the hierarchy also follow the above format.

Output

After following Technical Step 4 and unpacking the XML response, you will have the output from your request. This section describes the different outputs resulting from “Canned” test requests and “Live” requests. Both types of requests result in an XML response with the following tags:

Output	XML Tag
Type of Response	<CityStateLookupResponse...
ZIP Code Lookup Number	<ZipCode ID='#'>
ZIP Code of City or State	<Zip5>
City for Requested ZIP Code	<City>
State for requested ZIP Code	<State>

“Canned” Test Responses

For your test to be successful, the following responses should be returned *verbatim*. If any values were changes in your request, the following default error will occur:

```
<?xml version="1.0"?>
<CityStateLookupResponse>
  <ZipCode ID="0">
    <Error>
      <Number>-2147219040</Number>
      <Source>SOLServerTest;SOLServerTest.CallZipCodeDll</Source>
      <Description>This Information has not been included in this Test
      Server.</Description>
      <HelpFile></HelpFile>
      <HelpContext></HelpContext>
    </Error>
  </ZipCode>
</CityStateLookupResponse>
```

Although the input may be valid, the response will still raise this error, because those particular values have not been included in this test server. Refer to the *Errors* section for an explanation of any other returned errors.

Response to Valid Test Request #1

```
<?xml version="1.0"?>
<CityStateLookupResponse>
<ZipCode ID="0">
  <Zip5>90210</Zip5>
  <City>BEVERLY HILLS</City>
  <State>CA</State>
</ZipCode>
</CityStateLookupResponse>
```

Response to Valid Test Request #2

```
<?xml version="1.0"?>
<CityStateLookupResponse>
<ZipCode ID="0">
  <Zip5>20770</Zip5>
  <City>GREENBELT</City>
  <State>MD</State>
</ZipCode>
</CityStateLookupResponse>
```

Response to Valid Test Request #3

```
<?xml version="1.0"?>
<CityStateLookupResponse>
<ZipCode ID="0">
  <Zip5>21113</Zip5>
  <City>ODENTON</City>
  <State>MD</State>
</ZipCode>
</CityStateLookupResponse>
```

Response to Valid Test Request #4

```
<?xml version="1.0"?>
<CityStateLookupResponse>
<ZipCode ID="0">
  <Zip5>21032</Zip5>
  <City>CROWNSVILLE</City>
  <State>MD</State>
</ZipCode>
</CityStateLookupResponse>
```

Response to Valid Test Request #5

```
<?xml version="1.0"?>
<CityStateLookupResponse>
  <ZipCode ID="0">
    <Zip5>21117</Zip5>
    <City>OWINGS MILLS</City>
    <State>MD</State>
  </ZipCode>
</CityStateLookupResponse>
```

Response to Pre-defined Error Request: "Invalid ZIP Code"

```
<?xml version="1.0"?>
<CityStateLookupResponse>
  <ZipCode ID="0">
    <Error>
      <Number>-2147219403</Number>
      <Source>SQLServerTest;SQLServerTest.CallZipCodeDll</Source>
      <Description>Invalid Zip Code.</Description>
      <HelpFile></HelpFile>
      <HelpContext></HelpContext>
    </Error>
  </ZipCode>
</CityStateLookupResponse>
```

“Live” Responses

XML Output Example

```
<CityStateLookupResponse>  
  <ZipCode ID="0">  
    <Zip5>90210</Zip5>  
    <City>BEVERLY HILLS</City>  
    <State>CA</State>  
  </ZipCode>  
  <ZipCode ID="1">  
    <Zip5>20770</Zip5>  
    <City>GREENBELT</City>  
    <State>MD</State>  
  </ZipCode>  
</CityStateLookupResponse>
```