United States Postal Service®
Web Tool Kit User's Guide

A Technical Guide to

Track/Confirm

&

Track/Confirm Fields

Application Programming Interfaces

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

Version 3.1 (5/23/02)

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: icustomercare@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

Registered trademarks of the U.S. Postal Service	Trademarks of the U.S. Postal Service
Express Mail	Delivery Confirmation
First-Class Mail	Global Express Guaranteed
Global Priority Mail	Global Express Mail
Parcel Post	GXG
Priority Mail	International Parcel Post
ZIP + 4	Priority Mail Global Guaranteed
	Signature Confirmation
	ZIP Code

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

Adobe Acrobat is a trademark of Adobe Systems Incorporated.

Table of Contents

Introduction to the Track/Confirm API

The Track/Confirm API lets your customers determine the delivery status of their Priority Mail® and Package Services (Parcel Post™, Bound Printed Matter, Library Mail, and Media Mail) packages with Delivery Confirmation™. It will also provide tracking for Express Mail® and Global Express Mail™ right from your web site, without them having to go to the USPS web site. Additionally, the Track/Confirm API can be appended to your Intranet, allowing, for example, a customer service representative to answer customer queries about the status of their shipments. The API Server returns tracking and/or delivery confirmation information for packages requested by the client. The Track/Confirm API limits the data requested to ten (10) packages per transaction.

The Track/Confirm Field request (described starting on page 15) is identical to the Track/Confirm request except for the request name and the return information. Data returned still contains the detail and summary information, but this information is broken down into fields instead of having only one line of text.

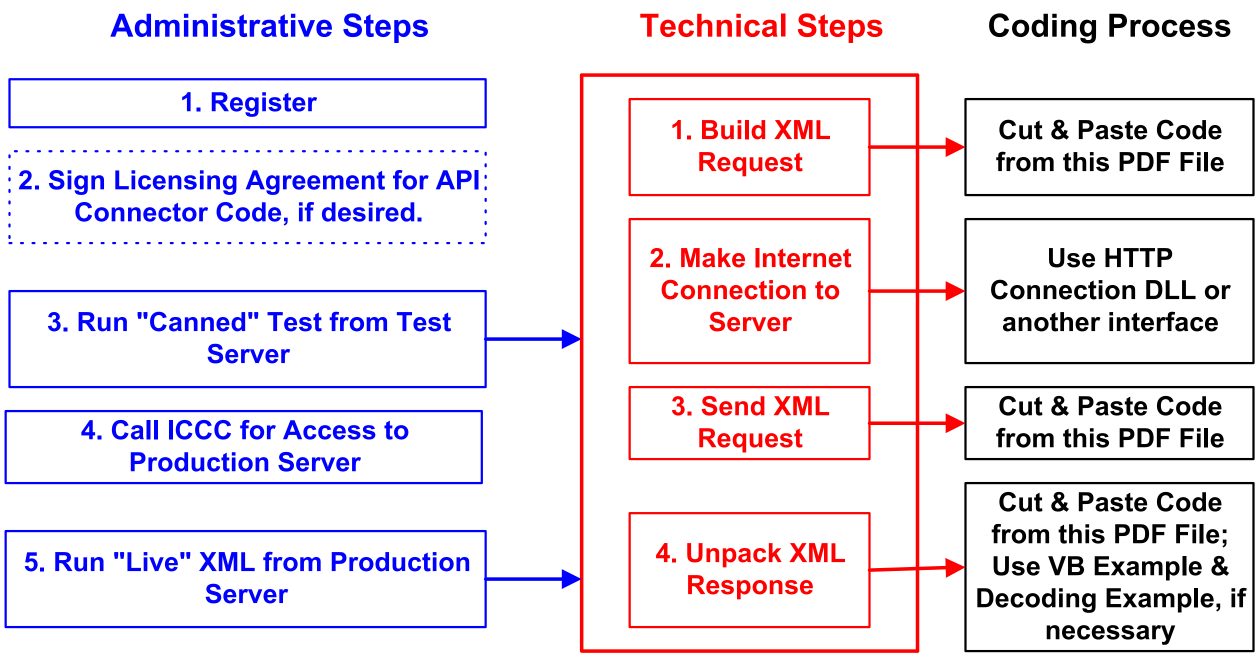
Do not attempt to parse returned data strings from the Track/Confirm API. If parsing is desired, use the Track/Confirm Fields API.

As shown below, implementing USPS Shipping APIs requires a series of *Administrative Steps*. The *Administrative Guide for APIs*, also available at www.uspswebtools.com, provides necessary information and procedures prior to installation.

The illustration also shows the *Technical Steps* required to run XML transactions for the Track/Confirm API 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. As each step is presented throughout this guide, appropriate portions of the illustration below will be repeated as a reference point in the implementation process.

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.


Administrative Steps
Technical Steps
Coding Process
1. Register
2. Sign Licensing Agreement for API Connector Code, if desired.
3. Run "Canned" Test from Test Server
4. Call ICCC for Access to Production Server
5. Run "Live" XML from Production Server
1. Build XML Request
2. Make Internet Connection to Server
3. Send XML Request
4. Unpack XML Response
Cut & Paste Code from this PDF File
Use HTTP Connection DLL or another interface
Cut & Paste Code from this PDF File
Cut & Paste Code from this PDF File; Use VB Example & Decoding Example, if necessary

User ID and Password Restrictions

The user ID and password that you have received are for you or your company to use in accordance with the Terms and Conditions of Use to which you agreed during the registration process. *This user ID and password are not to be shared with others outside your organization, nor are they 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.

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.

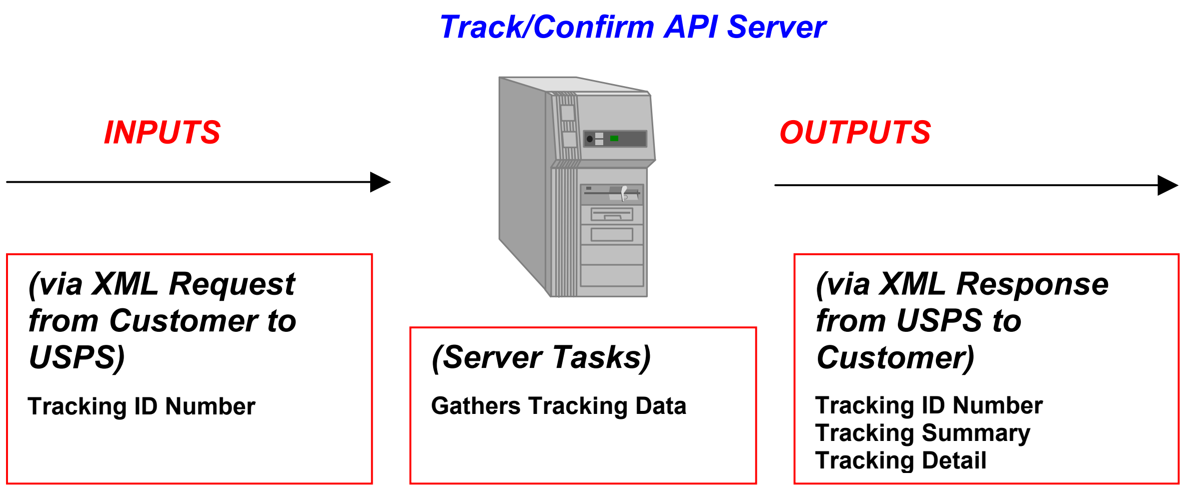
The documentation and sample code contained in the *Web Tool Kit User Guide* series may be reused and/or distributed to your customers or affiliates to generate awareness, encourage web tool use, or provide ease-of-use. However, it is your responsibility to ensure that your customers do not use your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

Note to Software Distributors: The User ID and password restrictions discussed above are intended for e-tailers that use the USPS Web Tools exclusively within their own web sites. If you plan to distribute software with the USPS Web Tools embedded, you must refer to the "Software Developers' Terms and Conditions of Use" available at www.uspswebtools.com.

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 icustomercare@usps.com.

Transaction Procedures

The illustration below shows the transactional flow of information to and from the USPS Track/Confirm API server.



Technical Steps

Step 1: Build the XML Request

1. Build XML Request → **Cut & Paste Code from this PDF File**

"Canned" Test Requests

For testing purposes, the only values in the test code in this section that you should change are the "USERID," "PASSWORD," and "SERVERNAME." Enter the user ID, password, and server name you received in the registration e-mail for testing. Your user ID and password never change, but the server name will change later when you send "live" requests. The "live" server name will be provided when the ICCC provides you with access to the production server. ***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

abc or T

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 two valid requests included in 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.

Valid Test Request #1

```
http://SERVERNAME/ShippingAPITest.dll?API=TrackV2&XML=
<TrackRequest USERID="xxxxxxxx" PASSWORD="xxxxxxxx">
  <TrackID ID="EJ958083578US"></TrackID>
</TrackRequest>
```

Valid Test Request #2

```
http://SERVERNAME/ShippingAPITest.dll?API=TrackV2&XML=
<TrackRequest USERID="xxxxxxxx" PASSWORD="xxxxxxxx">
  <TrackID ID="EJ958088694US"></TrackID>
</TrackRequest>
```

Pre-Defined Error Request

The pre-defined error in this request is using an invalid Tracking number in `<TrackID>`.

```
http://SERVERNAME/ShippingAPITest.dll?API=TrackV2&XML=
<TrackRequest USERID="xxxxxxxx" PASSWORD="xxxxxxxx">
  <TrackID ID="666666"></TrackID>
</TrackRequest>
```

"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.

Input	XML Tag	Values Allowed
Type of Request	`<TrackRequest…`	Input tag exactly as presented.
User ID	`…USERID="userid"…`	Use user ID provided with registration.
Password	`…PASSWORD="password">`	Use password provided with registration.
Package Tracking ID Number (up to 10 per transaction)	`<TrackID ID="#########">`	Must be alphanumeric.

XML Request

The "Live" XML request should be in the form:

```
<TrackRequest USERID="xxxxxxxx" PASSWORD="xxxxxxxx">
      <TrackID ID="EJ123456780US"/>
      <TrackID ID="EJ123456781US"/>
      <TrackID ID="12345"/>
</TrackRequest>
```

Perl Request

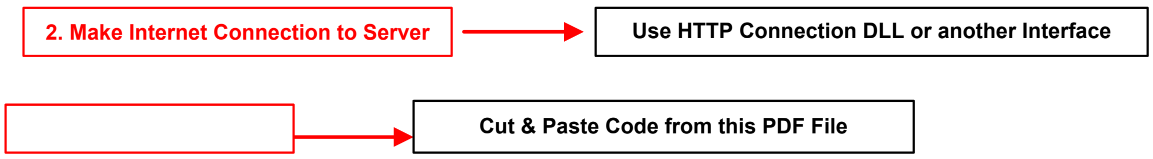
This sample requires the Perl XML parser and libwww-perl. Both are available from the Comprehensive Perl Archive Network (CPAN) at http://www.perl.com/CPAN/. Build the XML request with the parameters passed from the HTML form.

```
$query = new CGI;
$query->use_named_parameters(1);

$TrackReqDoc = new XML::DOM::Document;
$trackReqEl = $TrackReqDoc->createElement('TrackRequest');
$trackReqEl->setAttribute('USERID', 'xxxxxxxx');
$trackReqEl->setAttribute('PASSWORD', 'xxxxxxxx');
$TrackReqDoc->appendChild($trackReqEl);

for ($i = 1; $i <= 5; $i++) {
    $id = $query->param('TrackID' . $i);
    if (${id} ne "") {
        $trackIdEl = $TrackReqDoc->createElement('TrackID');
        $trackIdEl->setAttribute('ID', ${id});
        $trackReqEl->appendChild($trackIdEl);
    }
}
```

ASP Request

```
set xmlDoc = Server.CreateObject("MSXML.DOMDocument")

Set n = xmlDoc.createElement("TrackRequest")
n.setAttribute "USERID", "xxxxxxx"
n.setAttribute "PASSWORD", "xxxxxxx"

x=1
For i = 1 To 5
      if Request.Form("txtTrack" & i) <> "" then
            Set c = xmlDoc.createElement("TrackID")
            c.setAttribute "ID", Request.Form("txtTrack" & i)
            Call n.appendChild(c)
            x=x+1
      end if
Next

Call xmlDoc.appendChild(n)
```

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


2. Make Internet Connection to Server
Use HTTP Connection DLL or another Interface
Cut & Paste Code from this PDF File

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 server 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 three 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 Perl

Perl is recommended for UNIX systems. This sample requires the Perl XML parser and libwww-perl. Both are available from the Comprehensive Perl Archive Network (CPAN) at http://www.perl.com/CPAN/.

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 *"Canned"* test requests the code should read:

```
File = "/ShippingAPItest.dll?"
```

For *"Live"* requests the code should read:

```
File = "/ShippingAPI.dll?"
```

Input:

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

File = "/ShippingAPI.dll?"
xml = "API=TrackV2&XML=" & XMLSTRING

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

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

' Uncomment to use http GET request:
'File = File & xml
'hFile = HttpOpenRequest(hConnection, "GET", File, "HTTP/1.0", _
'  vbNullString, 0, 0, 0)

' Using http POST request:
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)
' end of code to use POST request

Call 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
        xmlStr = xmlStr & 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)
```

Using Perl

"SERVERNAME" indicates the name of the USPS web site to which you are connecting:

For *"Canned"* test requests the code should read:

```
$req = new HTTP::Request 'POST', 'http://SERVERNAME/ShippingAPItest.dll';
```

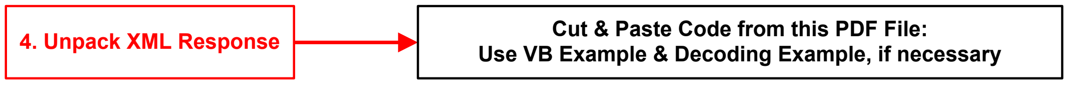
For *"Live"* requests the code should read:

```
$req = new HTTP::Request 'POST', 'http://SERVERNAME/ShippingAPI.dll';
```

Input:

```
print "content-type: text/html\n\n";
print $htmlBegin;

$ua = new LWP::UserAgent;
$req = new HTTP::Request 'POST', 'http://SERVERNAME/ShippingAPI.dll';
$req->content_type('application/x-www-form-urlencoded');
$req->content('API=TrackV2&XML=' . $TrackReqDoc->toString);
$response = $ua->request($req);
if ($response->is_success) {
    $resp = $response->content;
} else {
    print "<p>There was an error processing your request</p>\n";
    print $htmlEnd;
    exit 0;
}
```

Step 4: Unpack the XML Response


4. Unpack XML Response
Cut & Paste Code from this PDF File:
Use VB Example & Decoding Example, if necessary

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 DOMDocument
Dim elem As IXMLDOMElement
Dim node As IXMLDOMNode
Dim curID As String
Dim xmlStr As String

' assume xmlStr contains the XML string received from the USPS server
xmlDoc.validateOnParse = False
xmlDoc.loadXML (xmlStr)

txtResult.Text = ""
If xmlDoc.documentElement.nodeName = "Error" Then
   txtResult.Text = "Request-level error" & _
      xmlDoc.documentElement.Text
Else
   For Each elem In xmlDoc.documentElement.childNodes
      curID = elem.getAttribute("ID")
      ' display the Tracking ID
      txtResult.Text = txtResult.Text & curID
      For Each node In elem.childNodes
         If node.nodeName = "TrackSummary" Then
            ' display the summary
            txtResult.Text = txtResult.Text & vbCrLf & _
               "Package disposition: " & node.firstChild.nodeValue
         ElseIf node.nodeName = "TrackDetail" Then
            ' display the detail
            txtResult.Text = txtResult.Text & vbCrLf & _
               "Detail: " & node.firstChild.nodeValue
         End If
      Next node
      txtResult.Text = txtResult.Text & vbCrLf
   Next elem
End If
```

Using Perl

This sample requires the Perl XML parser and libwww-perl. Both are available from the Comprehensive Perl Archive Network (CPAN) at http://www.perl.com/CPAN/. Unpack the response and display as HTML.

```
$parser = new XML::DOM::Parser;
$trackRespDoc = $parser->parse($resp);
```

```
if ($trackRespDoc->getDocumentElement->getNodeName eq 'Error') {
    print "<p>There was an error processing your request</p>\n";
    print $htmlEnd;
    exit 0;
}

$trackInfoList = $trackRespDoc->getElementsByTagName('TrackInfo');
$n = $trackInfoList->getLength;

for ($i = 0; $i < $n; $i++) {
    print "                          <P>\n";
    $trackInfoNode = $trackInfoList->item($i);
    $id = $trackInfoNode->getAttribute('ID');
    print "                            <B>Tracking #: " . ${id};
    print "</B><BR>\n";
    print '                            <FONT SIZE="-1">' . "\n";
    $trackSummary = $trackInfoNode->getElementsByTagName('TrackSummary');
    $summary = $trackSummary->item(0)->getFirstChild->getNodeValue;
    $summary =~ tr/\n/ /s;
    @details = ();
    $trackDetailsList = $trackInfoNode->getElementsByTagName('TrackDetail');
    $l = $trackDetailsList->getLength;
    for ($j = 0; $j < $l; $j++) {
        $details[$j] = $trackDetailsList->item($j)->getFirstChild-
>getNodeValue;
        $details[$j] =~ tr/\n/ /s;
    }
    print "                            ";
    print $summary . "<BR>\n";
    if ($#details > 0) {
        print "                            <DL>\n";
        print "                              <DT>Details:\n";
        $LI = "                              <DD>";
        print $LI . join("\n" . $LI, @details) . "\n";
        print "                            </DL>\n";
    }
    print '                            </FONT>' . "\n";
    print "                          </P>\n";
}

print $htmlEnd;
```

Using ASP

```
Set xmlDoc = Server.CreateObject("MSXML.DOMDocument")

' assume resp contains the XML string received from the USPS server
xmlDoc.validateOnParse = False
xmlDoc.loadXML (resp)
Set nodeList = xmlDoc.getElementsByTagName("TrackInfo")

For i = 0 To nodeList.length - 1
      Set n = nodeList.Item(i)
      set curID = n.Attributes
      set x = curID.getNamedItem("ID")
      Response.Write("Tracking ID: " & x.nodeValue)
      For j = 0 To n.childNodes.length - 1
            Set e = n.childNodes.Item(j)
            If e.nodeName = "TrackSummary" Then
                  Response.Write("<P>")
                  Response.write("Summary: ")
                  Response.Write(e.firstChild.nodeValue)
                  Response.Write("</P>")
            ElseIf e.nodeName = "TrackDetail" Then
                  Response.Write("<P>")
                  Response.write("Detail: ")
                  Response.write(e.firstChild.nodeValue)
                  Response.Write("</p>")
            Else
                   'error
            End If
      Next 'j
Next 'i
```

Errors

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]

Error elements that are further down in the XML tree hierarchy also follow the above format.

An `<Error>` element may be returned at the top (response) level if there is a problem with the syntax of the request, or if a system error occurs. But if there is a problem with a specific tracking ID within the request, an `<Error>` element will be returned within the `<TrackInfo>` element that pertains to the specific tracking ID.

Since the Track/Confirm API allows you to submit multiple tracking IDs within a single request document, the response may contain a mix of tracking information and errors. For requests containing multiple tracking IDs, you need to check if there is an `<Error>` within a given `<TrackInfo>` element, as well as checking for an error at the top level.

```
<TrackResponse>
   <TrackInfo ID="EJ987654321US">
     <TrackSummary>Your item was delivered at 2:22 pm on October 28 in
     PROVIDENCE RI 02912.</TrackSummary>
```

```
      <TrackDetail>October 28 1:34 pm ARRIVAL AT UNIT PROVIDENCE RI
      02912</TrackDetail>
      <TrackDetail>October 28 10:54 am ARRIVAL AT UNIT PROVIDENCE RI
      02906</TrackDetail>
      <TrackDetail>October 27 7:12 pm ENROUTE 20770</TrackDetail>
      <TrackDetail>October 27 6:46 pm ACCEPT OR PICKUP 20770</TrackDetail>
   </TrackInfo>
       <TrackInfo ID="EJ888888888US">
      <TrackSummary> There is no record of that mail item. If it was mailed
      recently, It may not yet be tracked. Please try again later.
      </TrackSummary>
   </TrackInfo>
   <TrackInfo ID="bob">
      <TrackSummary> That's not a valid number. Please check to make sure you
      entered it correctly.</TrackSummary>
   </TrackInfo>
</TrackResponse>
```

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	`<TrackResponse>`
Package Tracking ID Number	`<TrackInfo ID="#######">`
Tracking Summary Information	`<TrackSummary>`
Tracking Detail Information (0, 1 or more may be returned)	`<TrackDetail>`

"Canned" Test Responses

For your test to be successful, the following responses to Valid Test Requests and Pre-defined Test Requests should be ***verbatim***. If any values were changed in your request, the following default error will be returned:

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

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

```
<TrackResponse>
      <TrackInfo ID="EJ958083578US">
          <TrackSummary>Your item was delivered at 8:10 am on June 1 in
          Wilmington DE 19801.</TrackSummary>
          <TrackDetail>May 30 11:07 am NOTICE LEFT WILMINGTON DE
          19801.</TrackDetail>
          <TrackDetail>May 30 10:08 am ARRIVAL AT UNIT WILMINGTON DE
          19850.</TrackDetail>
          <TrackDetail>May 29 9:55 am ACCEPT OR PICKUP EDGEWATER NJ
          07020.</TrackDetail>
      </TrackInfo>
</TrackResponse>
```

Response to Valid Test Request #2

```
<TrackResponse>
      <TrackInfo ID="EJ958088694US">
           <TrackSummary>Your item was delivered at 1:39 pm on June 1 in
           WOBURN MA 01815.</TrackSummary>
           <TrackDetail>May 30 7:44 am NOTICE LEFT WOBURN MA
           01815.</TrackDetail>
           <TrackDetail>May 30 7:36 am ARRIVAL AT UNIT NORTH READING MA
           01889.</TrackDetail>
           <TrackDetail>May 29 6:00 pm ACCEPT OR PICKUP PORTSMOUTH NH
           03801.</TrackDetail>
      </TrackInfo>
</TrackResponse>
```

Response to Pre-defined Error Request

```
<TrackResponse>
      <TrackInfo ID="666666">
           <TrackSummary>That is not a valid number.  Please check to make
           sure you entered it correctly.</TrackSummary>
           <TrackDetail>There is no information available for that
           package.</TrackDetail>
      </TrackInfo>
</TrackResponse>
```

"Live" Responses

The outputs, summary, and detail contain only the information from the USPS track/confirm web site. If there is no information on the site and it returns a message indicating that there is no information available, or that the tracking ID is invalid, this exact information is returned to the caller. This information is repeated for each of the tracking IDs supplied by the caller.

XML Output Example

```
<TrackResponse>
      <TrackInfo ID="E123456780US">
            <TrackSummary> Your item was delivered at 6:50 am on February 6
            in BARTOW FL 33830.</TrackSummary>
            <TrackDetail>February 6 6:49 am NOTICE LEFT BARTOW FL
            33830</TrackDetail>
            <TrackDetail>February 6 6:48 am ARRIVAL AT UNIT BARTOW FL
            33830</TrackDetail>
            <TrackDetail>February 6 3:49 am ARRIVAL AT UNIT LAKELAND FL
            33805</TrackDetail>
            <TrackDetail>February 5 7:28 pm ENROUTE 33699</TrackDetail>
            <TrackDetail>February 5 7:18 pm ACCEPT OR PICKUP
            33699</TrackDetail>
      </TrackInfo>
      <TrackInfo ID="E123456781US">
            <TrackSummary There is no record of that mail item. If it was
            mailed recently, It may not yet be tracked. Please try again
            later. </TrackSummary>
      </TrackInfo>
      <TrackInfo ID="12345">
            <TrackSummary> That's not a valid number. Please check to make
            sure you entered it correctly.</TrackSummary>
      </TrackInfo>
</TrackResponse>
```

Introduction to the Track/Confirm Fields API

The Track Field request is identical to the Track request except for the request name and the return information. Data returned still contains the detail and summary information, but this information is broken down into fields instead of having only one line of text. Up to 10 tracking IDs may be contained in each request input to the API server.

All tags are mandatory and must be present. Data for the tags is optional. All tags are case sensitive. The following tags are contained in the return information for both the detail and summary information in the Track/Confirm Fields API:

- EventTime - The time of the event; format: (h)h:mm am/pm
- EventDate - The date of the event; format: month (d)d, yyyy
- Event - The event type (e.g., Enroute)
- EventCity - The city where the event occurred
- EventState - The state where the event occurred (two-letter standard abbreviation)
- EventZIPCode - The 5-digit ZIP Code of the event
- EventCountry - The country where the event occurred
- FirmName - The company name if delivered to a company
- Name - The name of the person signing for delivery (if available)
- AuthorizedAgent - A True/False field to indicate the signer is an authorized agent.

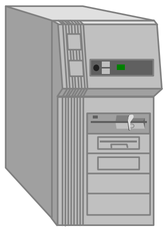
User ID and Password Restrictions

The user ID and password restrictions for this API are identical to those for the Track/Confirm API. Refer to page 2.

Transaction Procedures

The illustration on the following page shows the transactional flow of information to and from the USPS Track/Confirm Fields API server.

Track/Confirm Fields API Server



INPUTS →

(via XML Request from Customer to USPS)

Tracking ID Number

(Server Tasks)

Gathers Tracking Data

OUTPUTS →

(via XML Response from USPS to Customer)

Tracking ID Number	Tracking Detail
Tracking Summary	Event Time
Event Time	Event Date
Event Date	Event Type
Event Type	Event City
Event City	Event State
Event State	Event Zip Code
Event Zip Code	Event Country
Event Country	Firm Name
Firm Name	Name
Name	Authorized Agent
Authorized Agent	

Technical Steps

Step 1: Build the XML Request

1. Build XML Request → Cut & Paste Code from this PDF File

"Canned" Test Requests

For testing purposes, the only values in the test code in this section that you should change are the "USERID," "PASSWORD," and "SERVERNAME." Enter the user ID, password, and server name you received in the registration e-mail for testing. Your user ID and password never change, but the server name will change later when you send "live" requests. The "live" server name will be provided when the ICCC provides you with access to the production server. ***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

abc or T

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 two valid requests included in 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.

Valid Test Request #1

```
http://SERVERNAME/ShippingAPITest.dll?API=TrackV2&XML=
<TrackFieldRequest USERID="XXXXXXX" PASSWORD="XXXXXXX">
  <TrackID ID="EJ958083578US"></TrackID>
</TrackFieldRequest>
```

Valid Test Request #2

```
http://SERVERNAME/ShippingAPITest.dll?API=TrackV2&XML=
<TrackFieldRequest USERID="XXXXXXX" PASSWORD="XXXXXXX">
  <TrackID ID="EJ958088694US"/>
</TrackFieldRequest>
```

Pre-Defined Error Request (PIC not found)

```
http://SERVERNAME/ShippingAPITest.dll?API=TrackV2&XML=
<TrackFieldRequest USERID="XXXXXXX" PASSWORD="XXXXXXX">
  <TrackID ID="999999"></TrackID>
</TrackFieldRequest>
```

"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.

Input	XML Tag	Values Allowed
Type of Request	`<TrackFieldRequest…`	Input tag exactly as presented.
User ID	`…USERID="userid"…`	Use user ID provided with registration.
Password	`…PASSWORD="password">`	Use password provided with registration.
Package Tracking ID Number (up to 10 per transaction)	`<TrackID ID="#########">`	Must be alphanumeric.

XML Request

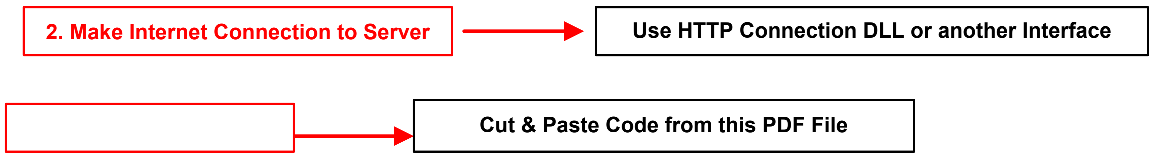
The “Live” XML request should be in the form:

```
<TrackFieldRequest USERID=”xxxxxxxx” PASSWORD=”xxxxxxxx”>
      <TrackID ID="EJ123456780US"/>
      <TrackID ID="EJ123456781US"/>
      <TrackID ID="12345"/>
</TrackFieldRequest>
```

ASP Request

```
set xmlDoc = Server.CreateObject("MSXML.DOMDocument")

Set n = xmlDoc.createElement("TrackFieldRequest")
n.setAttribute "USERID", "xxxxxxx"
n.setAttribute "PASSWORD", "xxxxxxx"

x=1
For i = 1 To 5
      if Request.Form("txtTrack" & i) <> "" then
            Set c = xmlDoc.createElement("TrackID")
            c.setAttribute "ID", Request.Form("txtTrack" & i)
            Call n.appendChild(c)
            x=x+1
      end if
Next

Call xmlDoc.appendChild(n)
```

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


2. Make Internet Connection to Server
Use HTTP Connection DLL or another Interface
Cut & Paste Code from this PDF File

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 server 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 on Microsoft platforms. Perl is recommended For UNIX systems, Perl is recommended.

- 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 *"Canned"* test requests the code should read:

```
File = "/ShippingAPItest.dll?"
```

For *"Live"* requests the code should read:

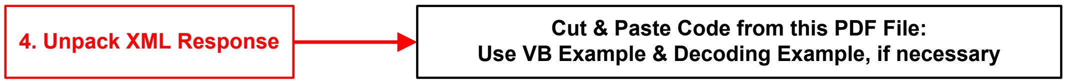
```
File = "/ShippingAPI.dll?"
```

Input:

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

File = "/ShippingAPI.dll?"
xml = "API=TrackV2&XML=" & XMLSTRING

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

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

' Uncomment to use http GET request:
```

```
'File = File & xml
'hFile = HttpOpenRequest(hConnection, "GET", File, "HTTP/1.0", _
'  vbNullString, 0, 0, 0)

' Using http POST request:
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)
' end of code to use POST request

Call 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
        xmlStr = xmlStr & 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


4. Unpack XML Response
Cut & Paste Code from this PDF File:
Use VB Example & Decoding Example, if necessary

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 DOMDocument
Dim nodeList As IDOMNodeList
Dim xmlDoc As New DOMDocument
Dim elem As IXMLDOMElement
Dim node As IXMLDOMNode, subNode As IXMLDOMNode
Dim curID As String
Dim xmlStr As String

' Assume xmlStr contains the XML string returned from the USPS server
xmlDoc.validateOnParse = False
xmlDoc.loadXML (xmlStr)

txtResult.Text = ""
If xmlDoc.documentElement.nodeName = "Error" Then
   txtResult.Text = "Request-level error" & _
      xmlDoc.documentElement.Text
Else
   For Each elem In xmlDoc.documentElement.childNodes
      curID = elem.getAttribute("ID")
      ' display the Tracking ID
      txtResult.Text = txtResult.Text & curID
      For Each node In elem.childNodes
         Select Case node.nodeName
            Case "TrackSummary"
               txtResult.Text = txtResult.Text & vbCrLf & _
                "Package disposition: " & node.firstChild.nodeValue
            Case "TrackDetail"
               txtResult.Text = txtResult.Text & vbCrLf & _
                  "Detail: " & node.firstChild.nodeValue
            Case "Error"
               txtResult.Text = txtResult.Text & vbCrLf & _
                  "Error: " & node.firstChild.nodeValue
         End Select
         ' display the leaf nodes
         For Each subNode In node.childNodes
            txtResult.Text = txtResult.Text & vbCrLf & _
               subNode.nodeName & ": " & subNode.Text
         Next subNode
      Next node
      txtResult.Text = txtResult.Text & vbCrLf
   Next elem
End If
```

Using ASP

```
Set xmlDoc = Server.CreateObject("MSXML.DOMDocument")

' Assume xmlStr contains the XML string returned from the USPS server
xmlDoc.validateOnParse = False
xmlDoc.loadXML (xmlStr)

If xmlDoc.documentElement.nodeName = "Error" Then
   Response.Write ("Request-level error: " & _
      xmlDoc.documentElement.Text)
Else
   For Each elem In xmlDoc.documentElement.childNodes
      curID = elem.getAttribute("ID")
      ' display the Tracking ID
      Response.Write ("<p><em>Tracking ID: " & curID & "</em>")
      For Each node In elem.childNodes
       Response.Write ("<p>")
         Select Case node.nodeName
            Case "TrackSummary"
               Response.Write ("Package disposition: ")
            Case "TrackDetail"
               Response.Write ("Detail: ")
            Case "Error"
               Response.Write ("Error: ")
         End Select
         ' display the leaf nodes
         Response.Write ("<ul>")
         For Each subNode In node.childNodes
            Response.Write ("<li>" & _
            subNode.nodeName & ": " & subNode.Text)
         Next 'subNode
         Response.Write ("</ul>")
      Next 'node
   Next 'elem

End If
```

Errors

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]

Error elements that are further down in the XML tree hierarchy also follow the above format.

An `<Error>` element may be returned at the top (response) level if there is a problem with the syntax of the request, or if a system error occurs. But if there is a problem with a specific tracking ID within the request, an `<Error>` element will be returned within the `<TrackInfo>` element that pertains to the specific tracking ID.

Since the Track/Confirm API allows you to submit multiple tracking IDs within a single request document, the response may contain a mix of tracking information and errors. For requests containing multiple tracking IDs, you need to check if there is an `<Error>` within a given `<TrackInfo>` element, as well as checking for an error at the top level.

Unlike the Track/Confirm API, the Track/Confirm Fields API returns an error if there is no information in the system for a given package.

```
<TrackResponse>
  <TrackInfo ID="01805213907000785963">
    <TrackSummary>
      <EventTime>2:11 pm</EventTime>
      <EventDate>April 24, 2001</EventDate>
      <Event>DELIVERED</Event>
      <EventCity>MADISON</EventCity>
      <EventState>WI</EventState>
      <EventZIPCode>53713</EventZIPCode>
      <EventCountry/>
      <FirmName></FirmName>
      <Name></Name>
      <AuthorizedAgent></AuthorizedAgent>
    </TrackSummary>
    <TrackDetail>
      <EventTime>10:00 pm</EventTime>
      <EventDate>April 21, 2001</EventDate>
      <Event>ACCEPTANCE</Event>
      <EventCity>HOUSTON</EventCity>
      <EventState>TX</EventState>
      <EventZIPCode>77009</EventZIPCode>
      <EventCountry/>
      <FirmName/>
      <Name/>
      <AuthorizedAgent/>
    </TrackDetail>
  </TrackInfo>
  <TrackInfo ID="EJ123456789US">
    <Error>
      <Number>-2147219302</Number>
      <Source>clsPostalTrack.ProcessErrorNode</Source>
      <Description>No record of that item</Description>
      <HelpFile/>
      <HelpContext/>
    </Error>
  </TrackInfo>
  <TrackInfo ID="-----------">
    <Error>
      <Number>-2147219297</Number>
      <Source>clsPostalTrack.ProcessErrorNode</Source>
      <Description>Invalid tracking id</Description>
      <HelpFile/>
      <HelpContext/>
    </Error>
  </TrackInfo>
</TrackResponse>
```

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	`<TrackResponse>`
Package Tracking ID Number	`<TrackInfo ID="#######">`
Tracking Summary Information	`<TrackSummary>`
Tracking Detail Information (0, 1 or more may be returned)	`<TrackDetail>`
The time of the event	`<EventTime>`
The date of the event	`<EventDate>`
The event type (e.g., Enroute)	`<Event>`
The city where the event occurred	`<EventCity>`
The state where the event occurred	`<EventState>`
The ZIP Code of the event	`<EventZIPCode>`
The country where the event occurred	`<EventCountry>`
The company name if delivered to a company	`<FirmName>`
The name of the persons signing for delivery (if available)	`<Name>`
True/False field indicating the person signing as an Authorized Agent	`<AuthorizedAgent>`

"Canned" Test Responses

For your test to be successful, the following responses to Valid Test Requests and Pre-defined Test Requests should be ***verbatim***. If any values were changed in your request, the following default error will be returned:

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

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

```
<TrackResponse>
      <TrackInfo ID="EJ958083578US">
          <TrackSummary>
            <EventTime>8:10 am</EventTime>
            <EventDate>June 1, 2001</EventDate>
            <Event>DELIVERED</Event>
            <EventCity>WILMINGTON</EventCity>
            <EventState>DE</EventState>
            <EventZIPCode>19801</EventZIPCode>
            <EventCountry/>
            <FirmName></FirmName>
            <Name></Name>
            <AuthorizedAgent></AuthorizedAgent>
         </TrackSummary>
         <TrackDetail>
            <EventTime>11:07 am</EventTime>
            <EventDate>May 30, 2001</EventDate>
            <Event>NOTICE LEFT</Event>
            <EventCity>WILMINGTON</EventCity>
            <EventState>DE</EventState>
            <EventZIPCode>19801</EventZIPCode>
            <EventCountry/>
            <FirmName/>
            <Name/>
            <AuthorizedAgent/>
        </TrackDetail>
        <TrackDetail>
            <EventTime>10:08 am</EventTime>
            <EventDate>May 30, 2001</EventDate>
            <Event>ARRIVAL AT UNIT</Event>
            <EventCity>WILMINGTON</EventCity>
            <EventState>DE</EventState>
            <EventZIPCode>19850</EventZIPCode>
            <EventCountry/>
            <FirmName/>
            <Name/>
            <AuthorizedAgent/>
        </TrackDetail>
        <TrackDetail>
            <EventTime>9:55 pm</EventTime>
            <EventDate>May 29, 2001</EventDate>
            <Event>ACCEPTANCE</Event>
            <EventCity>EDGEWATER</EventCity>
            <EventState>NJ</EventState>
            <EventZIPCode>07020</EventZIPCode>
            <EventCountry/>
            <FirmName/>
            <Name/>
            <AuthorizedAgent/>
        </TrackDetail>
      </TrackInfo>
</TrackResponse>
```

Response to Valid Test Request #2

```
<TrackResponse>
      <TrackInfo ID="EJ958088694US">
         <TrackSummary>
            <EventTime>1:39 pm</EventTime>
            <EventDate>June 1, 2001</EventDate>
            <Event>DELIVERED</Event>
            <EventCity>WOBURN</EventCity>
            <EventState>MA</EventState>
            <EventZIPCode>01815</EventZIPCode>
            <EventCountry/>
            <FirmName></FirmName>
            <AuthorizedAgent></AuthorizedAgent>
        </TrackSummary>
        <TrackDetail>
            <EventTime>7:44 am</EventTime>
            <EventDate>May 30, 2001</EventDate>
            <Event>NOTICE LEFT</Event>
            <EventCity>WOBURN</EventCity>
            <EventState>MA</EventState>
            <EventZIPCode>01815</EventZIPCode>
            <EventCountry/>
            <FirmName/>
            <Name/>
            <AuthorizedAgent/>
        </TrackDetail>
        <TrackDetail>
            <EventTime>7:36 am</EventTime>
            <EventDate>May 30, 2001</EventDate>
            <Event>ARRIVAL AT UNIT</Event>
            <EventCity>NORTH READING</EventCity>
            <EventState>MA</EventState>
            <EventZIPCode>01889</EventZIPCode>
            <EventCountry/>
            <FirmName/>
            <Name/>
            <AuthorizedAgent/>
        </TrackDetail>
        <TrackDetail>
            <EventTime>6:00 pm</EventTime>
            <EventDate>May 29, 2001</EventDate>
            <Event>ACCEPTANCE</Event>
            <EventCity>PORTSMOUTH</EventCity>
            <EventState>NH</EventState>
            <EventZIPCode>03801</EventZIPCode>
            <EventCountry/>
            <FirmName/>
            <Name/>
            <AuthorizedAgent/>
        </TrackDetail>
      </TrackInfo>
</TrackResponse>
```

Response to Pre-defined Error Request

```
<TrackResponse>
  <TrackInfo ID="999999">
    <Error>
      <Number>-2147219302</Number>
      <Source>clsPostalTrack.ProcessErrorNode</Source>
      <Description>No record of that item</Description>
      <HelpFile/>
      <HelpContext/>
    </Error>
  </TrackInfo>
```

```
</TrackResponse>
```

"Live" Responses

XML Output Example

```
<TrackResponse>
  <TrackInfo ID="01805213907042762274">
    <TrackSummary>
      <EventTime>12:12 pm</EventTime>
      <EventDate>May 21, 2001</EventDate>
      <Event>DELIVERED</Event>
      <EventCity>NEWTON</EventCity>
      <EventState>IA</EventState>
      <EventZIPCode>50208</EventZIPCode>
      <EventCountry/>
      <FirmName></FirmName>
      <Name></Name>
      <AuthorizedAgent></AuthorizedAgent>
    </TrackSummary>
    <TrackDetail>
      <EventTime>9:24 pm</EventTime>
      <EventDate>March 28, 2001</EventDate>
      <Event>ENROUTE</Event>
      <EventCity>DES MOINES</EventCity>
      <EventState>IA</EventState>
      <EventZIPCode>50395</EventZIPCode>
      <EventCountry/>
      <FirmName/>
      <Name/>
      <AuthorizedAgent/>
    </TrackDetail>
    <TrackDetail>
      <EventTime>10:00 pm</EventTime>
      <EventDate>March 27, 2001</EventDate>
      <Event>ACCEPTANCE</Event>
      <EventCity>BLAINE</EventCity>
      <EventState>WA</EventState>
      <EventZIPCode>98231</EventZIPCode>
      <EventCountry/>
      <FirmName/>
      <Name/>
      <AuthorizedAgent/>
    </TrackDetail>
  </TrackInfo>
</TrackResponse>
```