UPS Developer API Technologies for XML

The foundation technologies for UPS Developer APIs are the same underlying technologies in use on the Internet today. They include the Hypertext Transfer Protocol (HTTP) with Secure Sockets Layer 3 (SSL3) security and the Extensible Markup Language (XML). Because these technologies are critical to the Internet, most software development tools provide many powerful features that make it easy for application programs to use them. This section provides a quick introduction to the technologies and to the tools that can support them.

Hypertext Transfer Protocol and Secure Sockets Layer 3

Today's Internet supports a wide variety of applications including email, instant messaging, web browsing, and voice communications. All of these applications have a well-defined set of rules that determines how they communicate. Those rules are known as *protocols*. As an Internet application themselves, the UPS Developer APIs also rely on protocols to set the rules for their communications. The primary protocol for UPS Developer APIs is the hypertext transfer protocol, or HTTP.

HTTP is a relatively simple protocol. An application that wants to use a Developer API acts as an HTTP client. It sends its request to UPS as part of an HTTP POST message. UPS servers reply to each request using an HTTP response message.

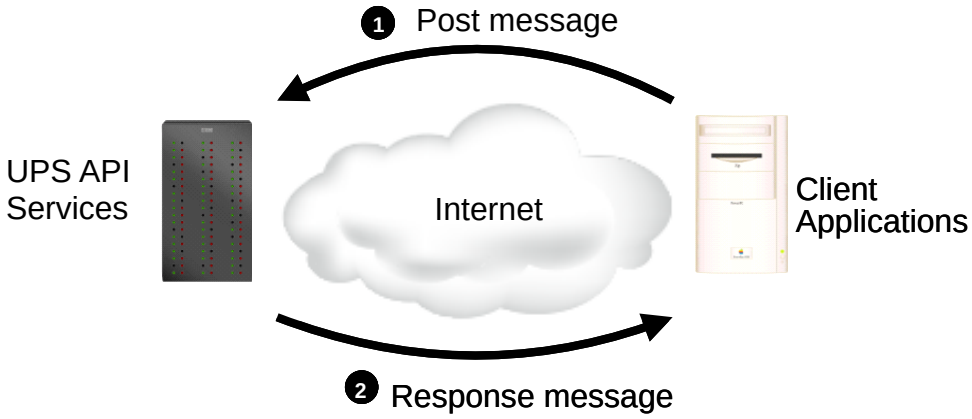


Figure 1. UPS Developer APIs accept requests from client applications in HTTP POST messages and reply to the requests with HTTP responses.

As part of the POST message, client applications indicate the type of content the message contains using an HTTP Content-Type header. For UPS Developer APIs, that content type should be application/x-www-form-urlencoded. (Presently, Version 1.1 is supported)

In many cases the UPS Developer APIs exchange information that should be kept private. To protect confidential information, the Developer APIs rely on the Secure Sockets Layer 3 (SSL3) protocol in addition to HTTP. When two systems

© 2010 United Parcel Service of America, Inc. All Rights Reserved. Confidential and Proprietary

communicate using SSL, the protocol creates a secure channel between them, and it encrypts all information that they exchange using this channel. The SSL protocol that Developer APIs use is the same protocol used to secure millions of on-line purchases on the web.

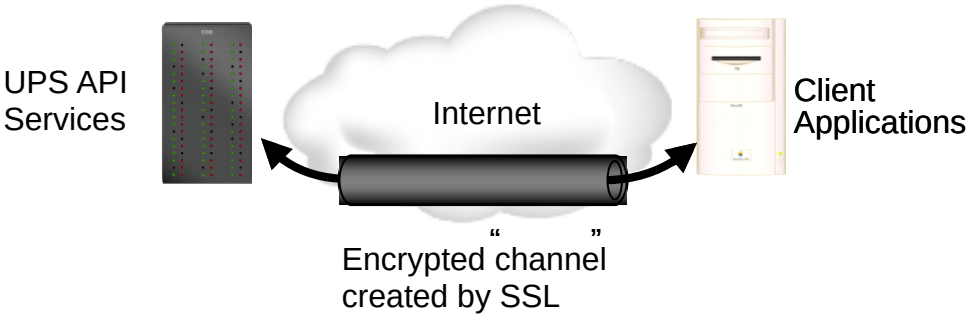
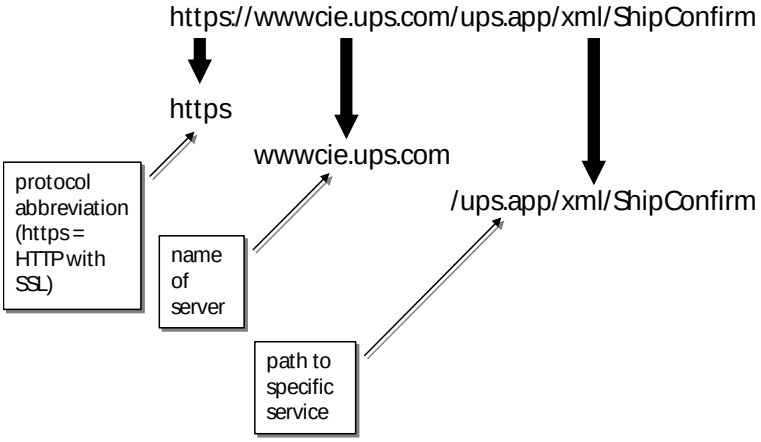


Figure 2: SSL creates a secure channel across a network and protects confidential communications using that channel.

Working with HTTP and SSL in Application Programs

UPS Developer APIs are not the only Internet services that rely on the HTTP and SSL protocols. Both protocols, in fact, were originally developed for web browsing, and they provide the foundation for the majority of Internet applications. Because HTTP and SSL are so common, software development tools make using these protocols very simple.

Because HTTP and SSL are commonly used for web browsing, software development tools often rely on the same notation and abbreviations as standard web browsers. In particular, most tools identify a specific service (such as a UPS Developer API) using a Uniform Resource Locator (URL). URLs begin with an abbreviation of the communication protocol. For UPS Developer APIs that abbreviation will always be "https" to indicate HTTP and SSL. The protocol abbreviation is followed by a colon, two slashes, and the name of a server. Additional information about the specific service can follow the server name; it is written like the path to a file in a directory, with slashes separating individual folders. Figure 3 shows how a URL combines these individual components.



© 2010 United Parcel Service of America, Inc. All Rights Reserved. Confidential and Proprietary

Figure 3. URLs identify communication protocols, servers, and specific services.

The subsections that follow provide brief introductions to using HTTP and SSL with various software development tools, including Microsoft Visual Studio, Java Standard and Enterprise Edition, and popular open source environments.

Microsoft Visual Studio

Visual Studio

The Microsoft Visual Studio environment includes many tools, classes, and libraries that simplify the use of HTTP and SSL protocols. A very convenient set of tools are included in the Microsoft XML Core Services (MSXML). The following code fragment shows how a Visual Basic program can send a request and retrieve the response. To keep the example as simple as possible, no exception handling is included in the figure. Production software should, of course, appropriately handle all exceptions.

```
' create the object that manages the communication
Dim oXMLHttp As XMLHTTP
Set oXMLHttp = New XMLHTTP

' prepare the HTTP POST request
oXMLHttp.open "POST", "https://www.server.com/path", False
oXMLHttp.setRequestHeader "Content-Type", _
                          "application/x-www-form-urlencoded"

' send the request
oXMLHttp.send requestString

' server's response will be available in oXMLHttp.responseXML
```

Figure 4. Visual Basic programs can use features of the MSXML tools to send and receive messages using HTTP and SSL.

Java Standard Edition and Enterprise Edition

Java

In a Java SE or EE environment, the URL class in the java.net package provides a convenient way to manage HTTP and SSL communications. Figure 5 contains a small code fragment that demonstrates the use of this class. As above, the fragment doesn't include exception handling that should be inherent in any production software.

```
import java.io.*;
import java.net.*;

URL url = new URL("https://www.server.com/path");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("POST");
conn.setRequestProperty("Content-Type",
                        "application/x-www-form-urlencoded");
conn.setDoOutput(true);
PrintWriter out = new PrintWriter(conn.getOutputStream());
out.println(requestString);
out.close();
BufferedReader in = new BufferedReader(
    new InputStreamReader(conn.getInputStream()));
/* server response is available by reading the in object */
```

Figure 5. The URL class from the java.net package is a convenient way for Java programs to use HTTP and SSL

© 2010 United Parcel Service of America, Inc. All Rights Reserved. Confidential and Proprietary

Open Source

Open Source

The libcurl project is an open source project that includes support for using HTTP and SSL protocols. The fragment in Figure 6 shows how to set up libcurl to send a request using HTTP and SSL. It relies on a callback function to accept the server's response.

```
CURL *curl;
CURLcode res;

/* prepare to send the request */
curl = curl_easy_init();
curl_easy_setopt(curl, CURLOPT_URL, https://www.server.com/path");
curl_easy_setopt(curl, CURLOPT_POSTFIELDS, requestString);
curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, fnCallback);

res = curl_easy_perform(curl);
```

Figure 6. The open source libcurl library simplifies the use of HTTP and SSL

Extensible Markup Language (XML)

The Extensible Markup Language (XML) is an international standard developed by the World Wide Web Consortium, the governing body for web standards and guidelines. XML provides a way to identify the structure of content within a document or, in the case of UPS Developer APIs, a message. Figure 7 shows how a simple XML message could describe a book.

```
<?xml version="1.0" encoding="UTF-8" ?>
<book>
    <title>
        HTTP Essentials: Protocols for Secure, Scaleable Web Sites
    </title>
    <author>
        <firstname>
            Stephen
        </firstname>
        <lastname>
            Thomas
        </lastname>
    </author>
    <publisher>
        John Wiley and Sons
    </publisher>
    <year>
        2001
    </year>
    <isbn>
        0-471-398233
```

Figure 7: XML identifies the structure of documents, as in this document describing a book.

As the figure illustrates, XML is a text-based format. XML messages contain regular text, though that text follows specific rules that XML defines. XML distinguishes different parts of a message with a label known as a *tag*. Tags in the example include <book>, <title>, <author>, <firstname>, etc. A tag can indicate the start of information if it begins with an angle bracket (<), and a tag can

© 2010 United Parcel Service of America, Inc. All Rights Reserved. Confidential and Proprietary

indicate the end of information if it begins with an angle bracket and a slash (</). A beginning tag, ending tag, and the information between the two make up an *element*. In this example the publisher element tells us that the publisher for the book is "John Wiley and Sons."

The figure also shows how an XML message follows a defined structure. Elements can be contained within other elements, as <firstname> and <lastname> are included within the <author> element. This structure indicates that firstname and lastname are "children" of the author "parent."

Although XML elements can contain almost any text, there are two special characters that cannot appear within an element. Those characters are the less-than sign (<) and the ampersand (&). Elements that include these contents must replace the special characters with "<" or "&" respectively. The name of a large telecommunications company, for example, would appear as "AT&T" within an XML message.

Because XML is a text-based format, software has to take extra steps to use XML with binary data. For example, some UPS services return binary image data such as the image of a shipping label. To include this data in an XML message, UPS converts it from binary to text format using an algorithm known as *Base64 encoding*. The "The Base16, Base32, and Base64 Data Encodings" standard, also known as RFC 3548, defines that algorithm.

Although XML messages consist of a series of text characters, most software development tools represent XML messages in a tree-like data structure. That representation clarifies the structure of the message. Figure 8 shows the same XML message as Figure 7, but it uses a more visual representation to highlight the message's structure.

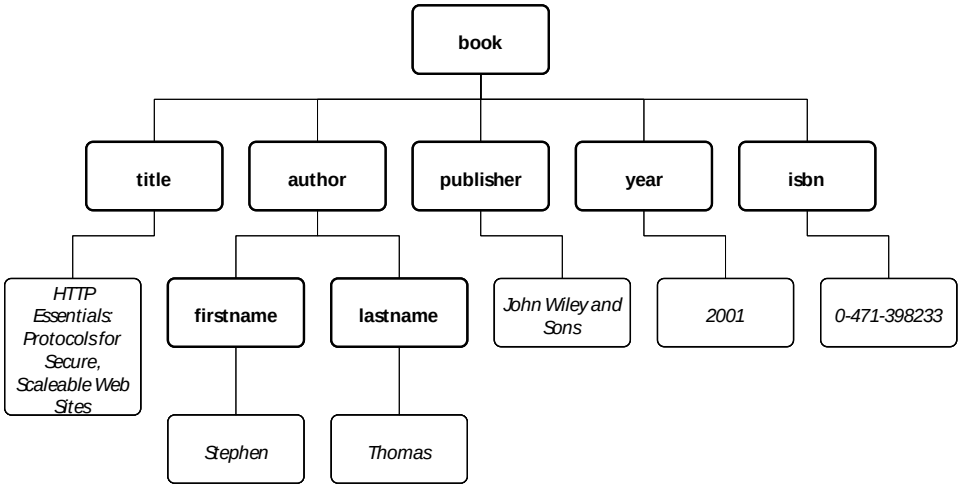


Figure 8. It is often convenient to show XML messages in a tree-like graph to highlight their structure.

The example of Figure 7 and Figure 8 is a relatively simple XML message with just a few elements. UPS Developer APIs rely on XML messages that are larger than this simple example, and it is difficult to show them in a pure graph like Figure 8 without losing legibility. Figure 9 shows an alternate way of depicting the tree-like structure of XML messages; it uses the same example as before.

© 2010 United Parcel Service of America, Inc. All Rights Reserved. Confidential and Proprietary

```
book
├─title
│  └─"HTTP Essentials: Protocols for Secure, Scaleable Web Sites"
├─author
│  ├─firstname
│  │  └─"Stephen"
│  └─lastname
│     └─"Thomas"
├─publisher
│  └─"John Wiley and Sons"
├─year
│  └─"2001"
└─isbn
   └─"0-471-398233"
```

Figure 9. It is possible to show the tree-like structure of XML messages using text.

A great deal of information on XML is available on the Internet. A good starting point for further research is the World Wide Web Consortium's main page on XML at http://www.w3.org/XML/.

Working with XML in Application Programs

The popularity of Extensible Markup Language means that nearly all development environments include sophisticated support for creating and interpreting XML messages. This subsection provides a brief introduction to working with XML in Microsoft, Java, and Open Source environments. The examples it contains are only representative approaches; many other approaches are possible.

Microsoft Visual Studio

Visual Studio

The Microsoft XML Core Services (MSXML) offers one way to work with XML documents in Visual Studio applications. Figure 10 shows one approach for creating an XML message in Visual Basic, and Figure 11 demonstrates how Visual Basic programs can interpret XML responses. To remain as simple as possible, the examples do not include any exception handling. Production software should, of course, provide full exception handling.

```
' Define a variable and initialize it to a new XML message
Dim dom
Set dom = New DOMDocument30

' Set properties of the variable
dom.async = False
dom.validateOnParse = False
dom.resolveExternals = False
dom.preserveWhiteSpace = True

' Identify the message as XML version 1.0
Set node = dom.createProcessingInstruction("xml", "version='1.0'")
dom.appendChild node
Set node = Nothing

' Create the root (book) element and add it to the message
Dim root
Set root = dom.createElement("book")
dom.appendChild root
```

© 2010 United Parcel Service of America, Inc. All Rights Reserved. Confidential and Proprietary

```
' Create child elements and add them to the root
Dim node
Set node = dom.createElement("title")
node.text = "HTTP Essentials: ..."
root.appendChild node
Set node = Nothing

Set node = dom.createElement("author")
Dim child
Set child = dom.createElement("firstname")
child.text = "Stephen"
node.appendChild child
Set child = Nothing
Set child = dom.createElement("lastname")
child.text = "Thomas"
node.appendChild child
root.appendChild node

' And so on
```

Figure 10. Visual Basic can create XML messages through the DOMDocument object.

```
' Define a variable to hold the parsed message
Dim dom As New DOMDocument30
dom.async = False
dom.validateOnParse = False
dom.resolveExternals = False
dom.preserveWhiteSpace = True

' Try to parse the message
If dom.loadXML(messageText) = False Then
    ' The text did not contain valid XML
End If

' Get the title information from the message
Dim node As IXMLDOMNode
Set node = dom.selectSingleNode("/title")
If node Is Nothing Then
    ' The message did not contain title information
Else
    ' Do something with node.text
End If
```

Figure 11. Visual Basic can read the contents of XML messages after parsing them with the DOMDocument object.

Visual Basic also has built-in classes to interpret Base64-encoded values. The `Convert.FromBase64String()` function converts from a Base64-encoded string to an array of 8-bit unsigned integers

Java Standard Edition and Enterprise Edition

Java

The Java API for XML Processing (JAXP) is the preferred approach for working with XML messages in Java. Figure 12 shows how a BookClass object can be converted into an XML message, a process that JAXP calls *marshalling*. Similarly, Figure 13 shows the reverse process, where an XML message is converted into a BookClass object. This reverse process is *unmarshalling*. As above, exception handling code is omitted from these examples to make them as clear as possible.

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import java.io.StringWriter;
```

© 2010 United Parcel Service of America, Inc. All Rights Reserved. Confidential and Proprietary

```
/* create the book object */
BookClass book = new BookClass("HTTP Essentials...",
    new AuthorClass("Stephen", "Thomas"),
    "John Wiley and Sons",
    "2001",
    "0-471-398233");

/* convert it to an XML string */
StringWriter writer = new StringWriter();
JAXBContext context = JAXBContext.newInstance(book.class);
Marshaller m = context.createMarshaller();
m.marshal(book, writer);
```

Figure 12. JAXP provides a convenient way for Java applications to create XML messages from Java objects.

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import java.io.StringReader;

/* xmlMessage contains the XML message */
StringReader reader = new StringReader(xmlMessage);
JAXBContext context = JAXBContext.newInstance(Book.class);
Unmarshaller u = context.createUnmarshaller();
BookClass book = (BookClass) u.unmarshal(reader);

/* access properties of the book */
System.out.println(book.getAuthor().getFirstname())
```

Figure 13. JAXP also gives Java applications a convenient way to retrieve elements from an XML message

Open Source

Open Source

As of this writing, the most popular open source tool for working with XML is the xerces project. That project includes C++, Java, and Perl implementations of an XML library. The xerces distribution includes extensive sample applications. The code fragment in Figure 14 demonstrates creating an XML message in C++ with the xerces library. Figure 15 shows sample code for accessing an element within an XML message. In both cases exception handling is not shown in order to keep the examples as simple as possible. Production software should always contain full exception handling.

```
DOMImplementation* impl =
    DOMImplementationRegistry::getDOMImplementation(X("Core"));

DOMDocument* doc = impl->createDocument(0, X("book"), 0);
DOMElement* rootElem = doc->getDocumentElement();

DOMElement* titleElem = doc->createElement(X("title"));
rootElem->appendChild(titleElem);

DOMText* titleVal = doc->createTextNode(X("HTTP Essentials..."));
titleElem->appendChild(titleVal);
```

Figure 14. The Xerces C++ library includes classes that can create XML messages.

```
XMLCh* TAG_book;
XMLCh* TAG_title;
xercesc::XercesDOMParser *parser;

XMLPlatformUtils::Initialize();
TAG_book  = XMLString::transcode("book");
TAG_title = XMLString::transcode("title");
```

© 2010 United Parcel Service of America, Inc. All Rights Reserved. Confidential and Proprietary

```
parser = new XercesDOMParser;
parser->parse(inputText);
DOMDocument* xmlDoc = parser->getDocument();
DOMElement* elementRoot = xmlDoc->getDocumentElement();
DOMNodeList* children = elementRoot->getChildNodes();
const XMLSize_t nodeCount = children->getLength();
for (XMLSize_t cnt = 0; cnt < nodeCount; ++cnt ) {
    DOMNode* currentNode = children->item(cnt);
    if( currentNode->getNodeType() &&
        currentNode->getNodeType() == DOMNode::ELEMENT_NODE ) {
        DOMElement* currentElement =
            dynamic_cast< xercesc::DOMElement* >( currentNode );
        if( XMLString::equals(currentElement->getTagName(),
                              TAG_title)) {
            /* do something with the title */
        }
    }
}
```

Figure 15. The Xerces C++ library also supports parsing XML messages to find individual elements.

© 2010 United Parcel Service of America, Inc. All Rights Reserved. Confidential and Proprietary