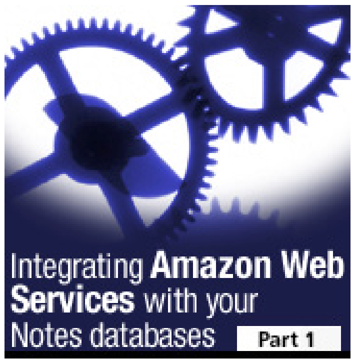


Level: Advanced
Works with: Notes/Domino
Updated: 03-Mar-2003

by
Ken
Yee

Long-time Amazon.com users will remember the "Eyes" service, which let you set up a keyword list of products that you were interested in. When new products arrived that match terms in your keyword list, you were sent an email notification with links to the new products along with descriptions that you could include on your Web site, which was useful if you were an Amazon Affiliate. With their quick growth, Amazon.com has since gotten rid of this service because it was too processor intensive for them to maintain.

Amazon.com recently created a Web service known as Amazon Web Services or AWS that allows you to query their search engine from your Web site. This Web service lets us create an application that is similar the Eyes service. You can integrate Amazon Web Services with a Notes application to create something that is even better than the Eyes service; the integrated solution provides a way to categorize the books using your own criteria, to mark irrelevant books, and to track when a book is no longer published. This technique is currently used to update the book list on the **Notes FAQ Web site**. You can, of course, apply this technique to other categories of Amazon's growing list of products.

In part 1 of this two-part article, we describe how to integrate Amazon Web Services (AWS) with a Notes application to query the Amazon.com search engine using the Notes Java API. We'll also describe how to create an agent that removes references to out-of-print books. In part 2 of this article, we'll show you how to reproduce this application using RDBMS and J2EE technologies. This article assumes that you are an experienced Notes application developer with Java programming skills. You can **download Amazon's Web Services SDK** directly from the Amazon.com Web site.

SOAP and Web services

There are many ways to communicate via Web services, but they are commonly accessed in one of two ways: XML/HTTP or Simple Object Access Protocol (SOAP). The XML/HTTP method uses a specially formatted URL for requests. Because the XML/HTTP access method uses a URL with encoded parameters to initiate the request, it limits you to the maximum length of a URL, which is roughly 1,024 characters (some servers support more, but this is a safe rule of thumb). The result you receive is XML/SOAP-encoded. You can transform the XML using XSLT or you can use an XML parser to read values from the XML. However, writing XML parser code can be tedious.

The SOAP technique for accessing Web services utilizes a Web Services Description Language (WSDL) definition provided by the Web services provider. The WSDL definition is required if you publish a Web service and is always available via a URL as you'll see later in this article. In our case, Amazon Web Services provides the WSDL definition. The WSDL definition describes how to format requests and how to read replies, and it tells you which requests are available. WSDL appears to add another complicated layer on top of Web services, but what it actually provides is a bridge needed to generate "wrappers" to access the Web services as though the service resided on your own machine.

Wrappers for calling Web services can be generated in any language as can be seen by the plethora of languages that support Web services in a friendly way, including Java, C, VB, PHP, Perl, TCL, and so on. In addition, the information sent in a "SOAP request" has no size limit because it is sent in the body of a Web request instead of the URL; this allows you to make complicated requests to the Web service because there is no size limit in the body of a Web request.

Amazon Web Services

You can find out more information about Amazon Web Services at the **Amazon.com Web Services page**. You'll find the SDK with sample code for Java, Perl, PHP, and VB. Documentation for using the SDK is included with the kit. There is a discussion forum for developers which Amazon monitors. You need to register for your developer's token before you can make any calls to the Amazon Web Services APIs. The developer token is used to provide referral credit to Amazon Associates when customers use the Amazon Associate's Web site to add items to the customers' Amazon orders; if you use Amazon's XSLT service to translate their XML results into HTML, the token is automatically embedded in the returned URLs, so you can get your referral fees.

Amazon has exposed a lot of functionality with their Web services implementation, and it is a good example of what Web services can be used for. Not only do they provide calls to their search engine, but they also provide the ability to manage an Amazon user's shopping cart, wish lists, and wedding registry. In fact, the intention of Web services is to allow Amazon Associates (their paid referrer program) a more flexible way to build Web storefronts to help sell more Amazon products. That's why it is such a good example of a Web service: It helps the Web service provider increase profits.

For this article, we're interested in the Amazon search engine Web Services API. Before we get started, you need to do the following:

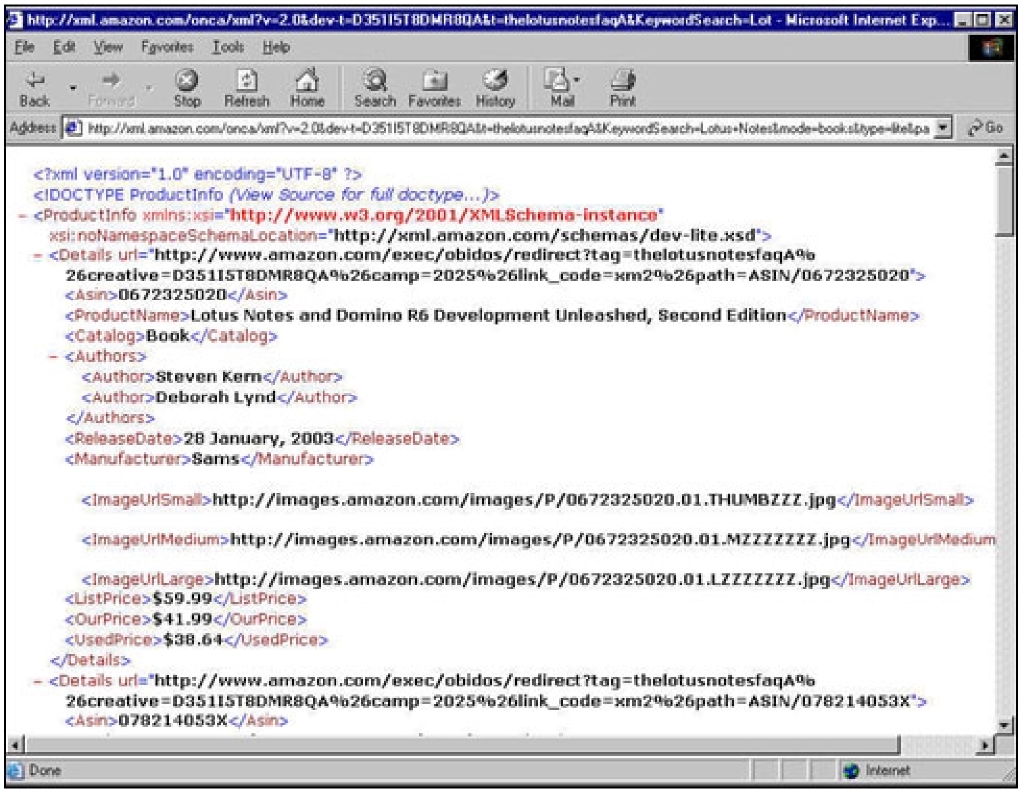
- Install the Amazon Web Service. Refer to the documentation available with Amazon Web Service SDK for more information.
- For test purposes, create a Notes application for use with AWS.

To use the Amazon search engine, the first thing you have to specify in your Notes application is something called the mode; this is equivalent to the upper tabs—books, music, tools, and so on—that you see on the Amazon.com home page. You can then search based on one of the following: Keyword, Browse Node (product category, for example, if mode is set to books, you can search science fiction), ASIN, ISBN, UPC, Author, Musician, Actor, Director, Manufacturer, ListMania!, and Similarity (what customers also bought). For the BetterEyes service that we're building, we use the Keyword search.

XML support in Notes/Domino 6

Notes/Domino 6 includes XML support, unlike Notes/Domino R5 which required add-on toolkits for XML support. We'll take a look at how our BetterEyes service can be done using the Notes XML support and then look at why we should use SOAP instead. For an example of the XML/HTTP that can be retrieved from Amazon Web Services, use Internet Explorer 5.5 or later to view **this URL**.

You receive a nicely formatted XML result that you can browse easily by clicking the + and - to expand and collapse parts of the hierarchy.


http://xml.amazon.com/onca/xml?v=2.0&dev-t=D351I5T8DMR8QA&t=thelotusnotesfaqA&KeywordSearch=Lot - Microsoft Internet Exp...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address http://xml.amazon.com/onca/xml?v=2.0&dev-t=D351I5T8DMR8QA&t=thelotusnotesfaqA&KeywordSearch=Lotus+Notes&mode=books&type=lite&pa
Go
Done
Internet

```
  <?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE ProductInfo (View Source for full doctype...)>
- <ProductInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://xml.amazon.com/schemas/dev-lite.xsd">
  - <Details url="http://www.amazon.com/exec/obidos/redirect?tag=thelotusnotesfaqA%
      26creative=D351I5T8DMR8QA%26camp=2025%26link_code=xm2%26path=ASIN/0672325020">
      <Asin>0672325020</Asin>
      <ProductName>Lotus Notes and Domino R6 Development Unleashed, Second Edition</ProductName>
      <Catalog>Book</Catalog>
    - <Authors>
        <Author>Steven Kern</Author>
        <Author>Deborah Lynd</Author>
      </Authors>
      <ReleaseDate>28 January, 2003</ReleaseDate>
      <Manufacturer>Sams</Manufacturer>

        <ImageUrlSmall>http://images.amazon.com/images/P/0672325020.01.THUMBZZZ.jpg</ImageUrlSmall>

        <ImageUrlMedium>http://images.amazon.com/images/P/0672325020.01.MZZZZZZZ.jpg</ImageUrlMedium

        <ImageUrlLarge>http://images.amazon.com/images/P/0672325020.01.LZZZZZZZ.jpg</ImageUrlLarge>
      <ListPrice>$59.99</ListPrice>
      <OurPrice>$41.99</OurPrice>
      <UsedPrice>$38.64</UsedPrice>
    </Details>
  - <Details url="http://www.amazon.com/exec/obidos/redirect?tag=thelotusnotesfaqA%
      26creative=D351I5T8DMR8QA%26camp=2025%26link_code=xm2%26path=ASIN/078214053X">
      <Asin>078214053X</Asin>
```

Using Notes/Domino 5 or 6, you can query Amazon.com using the Personal Web Navigator and this snippet of code in a Notes LotusScript agent:

```
Dim filename as String
filename$ = "c:\temp\amazon.xml"
Dim NURLdb As New NotesDatabase("","") ' To reference the Internet browser database
Dim Nwebdoc As NotesDocument
If NURLdb.OpenURLDb Then
    Dim filename As String
    ' download the XML encoded WS reply
    If (False) Then
    theURL = "
    http://xml.amazon.com/onca/xml?v=2.0&dev-t=webservices-20&t=thelotusnotesfaqA&KeywordSearch=Lotus+
    Notes&mode=books&type=lite&page=1&f=xml"
        Set Nwebdoc=NURLdb.getdocumentbyurl(theUrl, True, False)
        ' get XML from attachment
        Dim Item As NotesItem
        Dim Object As NotesEmbeddedObject
        Set Item = Nwebdoc.GetFirstItem("$FILE")
        Dim ItemName As String
        ItemName = Item.Values(0)
        Set Object = Nwebdoc.GetAttachment(ItemName)
        Call Object.ExtractFile(filename$)
    End If
End If
```

When Amazon sends the XML reply back, it's stored as an attachment in the Personal Web Navigator database. You have to save the attachment to your file system, so you can parse the XML from the reply. Notes/Domino 6 has two built-in XML parsers: SAX and DOM. The SAX parser uses less memory because it is a callback-style API, so it doesn't keep the entire XML document in memory. The DOM parser is a node-searching API, but it requires that the entire XML document be kept in memory. The DOM parser is slightly easier to use, so we'll focus on using that. For more information about the SAX and DOM parsers, see the *LDD Today* article, "**LotusScript: XML classes in Notes/Domino 6**."

The following code snippet prints the XML results from an Amazon Web Services query:

```
    ' open XML file as a stream
    Dim xml_in As NotesStream
    Set xml_in=session.CreateStream
    If xml_in.Open(filename$) Then
        If xml_in.Bytes = 0 Then
            Messagebox filename$ + " is empty"
        End If

        ' create a temporary output file because it's not a pipelined XML operation
        Dim xml_out As NotesStream
        Dim outfilename as String
        outfilename$ = "c:\temp\amazon.tmp" ' create output file
        Set xml_out=session.CreateStream
        If Not xml_out.Open(outfilename$) Then
            Messagebox "Cannot create " & outfilename$,, "TXT file error"
            Exit Sub
        End If
        xml_out.Truncate

        ' parse using DOM parser
        Set domParser=session.CreateDOMParser(xml_in, xml_out)

        On Error Goto DumpDOMLog
        DOMParser.ExitOnFirstFatalError = True
        DOMParser.InputValidationOption = 0 ' disable validation of problematic Amazon XML results
        domParser.Process
        On Error Goto 0

        Dim docNode As NotesDOMDocumentNode
        Set docNode = domParser.Document

        ' last child at root is ProductInfo
        Dim productinfoNode As NotesDOMNode
        Set productinfoNode = docNode.LastChild
        ' iterate through all the Details children
        Dim detailsNode As NotesDOMNode
        Set detailsNode = productinfoNode.FirstChild
        While (Not detailsNode.IsNull)
            Dim productname As String
            Dim publisher As String
            productname = ""
            Dim infoNode As NotesDOMNode
            Set infoNode = detailsNode.FirstChild
            While (Not infoNode.IsNull)
                If (infoNode.NodeName = "ProductName") Then
                    productname = infoNode.FirstChild.NodeValue
                Elseif (infoNode.NodeName = "Manufacturer") Then
                    publisher = infoNode.FirstChild.NodeValue
                End If
                Set infoNode = infoNode.NextSibling
            Wend
            If (productname <> "") Then
                Messagebox "Found " + productname + " from " + publisher
            End If
            Set detailsNode = detailsNode.NextSibling
        Wend
    End If
    Exit Sub

DumpDOMLog:
    Msgbox "DOMParser error: " & DOMParser.log
    Exit Sub
```

As you can see, it's relatively complicated to create the code that walks the XML document to retrieve the information that you need. The XML input validation was turned off because Amazon's XML reply had a problem with an invalid URL according to the DOMParser error log; turning off validation is necessary with a lot of XML parsing because of badly generated XML. In addition, you have to understand the XML schema enough to know which levels to look at for the information you want. In the previous example, the author information is in a lower level node. You also have to understand how the nodes are named and to spell them correctly (there is no error checking if you get them wrong).

In the next section, we'll see how a SOAP wrapper not only makes this easier but also more intuitive and safer.

Axis SOAP wrapper

As mentioned earlier, most languages have a SOAP wrapper generator. These generators take a WSDL description and generate a wrapper object so that using the Web service looks like it is part of the language. **Click here** to see what Amazon's Web Service WSDL description looks like. This is the latest version of Amazon's Web Services WSDL description. You can save it as a text file named AmazonWebServices.wsdl.

In Java, the de-facto SOAP wrapper has come out of the Apache Web server project. It's named Axis, and you can download it from the **Apache Axis Web site**. Axis is used in various forms in quite a few tools, among them IBM's Web Services Toolkit and Borland's JBuilder. Although we're only going to use it to generate a SOAP wrapper to talk to a Web service, it can also generate wrappers so that you can serve Java classes as Web services.

After you download Axis, follow the instructions provided on the Web site to install it. After you install Axis, you can tell it to generate wrapper classes for Amazon Web Services using this command line:

```
java java org.apache.axis.wsdl.WSDL2Java AmazonWebServices.wsdl --output build/axis --verbose --package com.amazon.soap.axis
```

This command generates Java files that will compile into the com.amazon.soap.axis Java package. This is an example of the code that it generates for the ProductInfo XML node:

```
/**
 * ProductInfo.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis WSDL2Java emitter.
 */

package com.amazon.soap.axis;

public class ProductInfo implements java.io.Serializable {
        private java.lang.String totalResults;
        private java.lang.String listName;
        private com.amazon.soap.axis.Details[] details;

        public ProductInfo() {
        }

        public java.lang.String getTotalResults() {
                return totalResults;
        }

        public void setTotalResults(java.lang.String totalResults) {
                this.totalResults = totalResults;
        }

        public java.lang.String getListName() {
                return listName;
        }
```

```
    public void setListName(java.lang.String listName) {
        this.listName = listName;
    }

    public com.amazon.soap.axis.Details[] getDetails() {
        return details;
    }

    public void setDetails(com.amazon.soap.axis.Details[] details) {
        this.details = details;
    }

    private java.lang.Object __equalsCalc = null;
    public synchronized boolean equals(java.lang.Object obj) {
        if (!(obj instanceof ProductInfo)) return false;
        ProductInfo other = (ProductInfo) obj;
        if (obj == null) return false;
        if (this == obj) return true;
        if (__equalsCalc != null) {
            return (__equalsCalc == obj);
        }
        __equalsCalc = obj;
        boolean _equals;
        _equals = true &&
            ((totalResults==null && other.getTotalResults()==null) ||
            (totalResults!=null &&
            totalResults.equals(other.getTotalResults()))) &&
            ((listName==null && other.getListName()==null) ||
            (listName!=null &&
            listName.equals(other.getListName()))) &&
            ((details==null && other.getDetails()==null) ||
            (details!=null &&
            java.util.Arrays.equals(details, other.getDetails())));
        __equalsCalc = null;
        return _equals;
    }

    private boolean __hashCodeCalc = false;
    public synchronized int hashCode() {
        if (__hashCodeCalc) {
            return 0;
        }
        __hashCodeCalc = true;
        int _hashCode = 1;
        if (getTotalResults() != null) {
            _hashCode += getTotalResults().hashCode();
        }
        if (getListName() != null) {
            _hashCode += getListName().hashCode();
        }
        if (getDetails() != null) {
            for (int i=0;
                i<java.lang.reflect.Array.getLength(getDetails());
                i++) {
                java.lang.Object obj = java.lang.reflect.Array.get(getDetails(), i);
                if (obj != null &&
                    !obj.getClass().isArray()) {
                    _hashCode += obj.hashCode();
                }
            }
        }
        __hashCodeCalc = false;
```

```
            return _hashCode;
        }

        // Type metadata
        private static org.apache.axis.description.TypeDesc typeDesc =
            new org.apache.axis.description.TypeDesc(ProductInfo.class);

        static {
            org.apache.axis.description.FieldDesc field = new org.apache.axis.description.ElementDesc();
            field.setFieldName("totalResults");
            field.setXmlName(new javax.xml.namespace.QName("", "TotalResults"));
            field.setXmlType(new javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema", "string"));
            typeDesc.addFieldDesc(field);
            field = new org.apache.axis.description.ElementDesc();
            field.setFieldName("listName");
            field.setXmlName(new javax.xml.namespace.QName("", "ListName"));
            field.setXmlType(new javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema", "string"));
            typeDesc.addFieldDesc(field);
            field = new org.apache.axis.description.ElementDesc();
            field.setFieldName("details");
            field.setXmlName(new javax.xml.namespace.QName("", "Details"));
            field.setXmlType(new javax.xml.namespace.QName("urn:PI/DevCentral/SoapService", "DetailsArray"));
            typeDesc.addFieldDesc(field);
        };

        /**
        * Return type metadata object
        */
        public static org.apache.axis.description.TypeDesc getTypeDesc() {
            return typeDesc;
        }

        /**
        * Get Custom Serializer
        */
        public static org.apache.axis.encoding.Serializer getSerializer(
            java.lang.String mechType,
            java.lang.Class _javaType,
            javax.xml.namespace.QName _xmlType) {
            return
            new org.apache.axis.encoding.ser.BeanSerializer(
            _javaType, _xmlType, typeDesc);
        }

        /**
        * Get Custom Deserializer
        */
        public static org.apache.axis.encoding.Deserializer getDeserializer(
            java.lang.String mechType,
            java.lang.Class _javaType,
            javax.xml.namespace.QName _xmlType) {
            return
            new org.apache.axis.encoding.ser.BeanDeserializer(
            _javaType, _xmlType, typeDesc);
        }

}
```

It looks relatively complicated, but it basically allows simple access to the XML results by making the XML ProductInfo node's attributes look like Java String objects and XML nodes underneath look like Java arrays of objects. The wrapper also makes the object "serializable," so you can transmit the XML nodes to Java programs over a network or save the XML nodes to a file.

After you generate the wrapper classes, you can create a simple program to call Amazon Web Services. Save the following code to a file named AmazonUpdate.java:

```
import com.amazon.soap.axis.*;
public class AmazonUpdate
{
	public static void main(String[] args) throws Exception
	{
		AmazonSearchService service = new AmazonSearchServiceLocator();
		AmazonSearchPort port = service.getAmazonSearchPort();
		KeywordRequest request = new KeywordRequest();

		request.setKeyword(java.net.URLEncoder.encode("Lotus Notes"));
		request.setMode("books");
		request.setTag("");
		request.setType("lite");
		request.setDevtag("");
		request.setPage("1");

		ProductInfo result = null;
		try {
			result = port.keywordSearchRequest(request);
		} catch (Exception e) {
			e.printStackTrace();
		}
		Details[] details = result.getDetails();
		for (int i = 0; i < details.length; i++) {
			// read results out
			String resultTitle = details[i].getProductName();
			System.out.println("Title #" + i + " is " + resultTitle);
		}
	}
}
```

Now you can compile this program and the SOAP wrapper by entering the following command:

```
javac AmazonUpdate.java build/axis/com/amazon/soap/axis/*.java
```

Run the program with the following command:

```
java -classpath build/axis/com/amazon/soap/axis;%CLASSPATH% AmazonUpdate
```

and it will print the titles of the books in the first page of results. Note how you don't even need to know much about Amazon Web Services. You don't even have to specify the URL to access Amazon Web Services because that information is embedded in the WSDL file. You'll also have Java's type/method checking to ensure that you don't call the Amazon Web Services API incorrectly or interpret the results with the wrong data type.

AmazonUpdate Java program

Now that we have a basic skeleton that can call Amazon Web Services, we can work on a full application that will do what we need. One unfortunate bug in Amazon's Web Services API is that the total result count is not correct. Each Web services call returns at most ten results. What we have to do is loop until we have less than ten results or until there is an error. The loop looks like this:

```
boolean fKeepgoing = true;
int realcount = 0;
int page = 0;
do {
	page++;
	request.setPage("" + page);

	ProductInfo result = null;
	try {
		result = port.keywordSearchRequest(request);
```

```
        } catch (Exception e) {
            org.apache.axis.AxisFault afe = null;
            if (e instanceof org.apache.axis.AxisFault) {
                afe = (org.apache.axis.AxisFault) e;
            }
            if ((afe == null) || (afe.getFaultString().indexOf("Bad Request") < 0)) {
                // Amazon's results "end" when the page you request gets a request error or if less than 10 results
                are returned
                e.printStackTrace();
            }
            fKeepgoing = false;
            break;
        }

        Details[] details = result.getDetails();
        for (int i = 0; i < details.length; i++) {
            // values that will be passed to tracker
            String resultTitle = null;
            String resultISBN = null;
            String resultASIN = null;
            String resultAuthors = null;
            String resultPublisher = null;

            // read results out
            resultTitle = details[i].getProductName();
            resultASIN = details[i].getAsin();
            if (details[i].getIsbn() != null) {
                resultISBN = details[i].getIsbn();
            }
            String authors[] = details[i].getAuthors();
            String authorlist = "";
            if (authors != null) {
                for (int j = 0; j < authors.length; j++) {
                if (j > 0) {
                    authorlist += ", ";
                }
                authorlist += authors[j];
                }
            }
            resultAuthors = authorlist;
            if (details[i].getManufacturer() != null) {
                resultPublisher = details[i].getManufacturer();
            }

            // give results to tracker
            tracker.updateBook(
                resultTitle,
                resultASIN,
                resultISBN,
                resultAuthors,
                resultPublisher);

            // increment result counter
            realcount++;
        }

        if (details.length < 10) {
            fKeepgoing = false;
        }
 } while (fKeepgoing);
```

BookUpdate interface

You may be wondering what the tracker.updateBook() call is for. It's a call to a Java interface that lets us update our database of books that we'll be tracking. By declaring this as an interface, we can use different classes that dump the information, so we can debug the application or write information to a Notes database or to a relational database or other datastore.

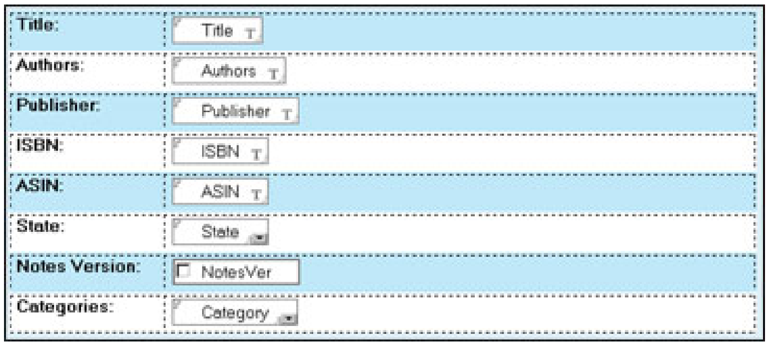
```
public interface BookTracker
{
    public void updateBook(String title, String ASIN, String ISBN, String authors, String publisher, String rating, int
    numreviews) throws Exception;
}
```

Initial debugging: Dumping to standard output
As you can imagine, the debugging class is very simple because all it does is print the information to the standard output stream which prints to the command window from which you run the Java application. Here's the class:

```
public class DumpBookTracker implements BookTracker {
    // update book from amazon query
    public void updateBook(
        String title,
        String ASIN,
        String ISBN,
        String authors,
        String publisher) {
        System.out.println(title);
        System.out.println("; ASIN: " + ASIN);
        if (ISBN != null) {
        System.out.println("; ISBN: " + ISBN);
    }
        if (publisher != null) {
        System.out.println("; Publisher: " + publisher);
    }
        if (authors != null) {
        System.out.println("; Authors: " + authors);
        }
    }
}
```

Integrating with Notes: Forms and views

Now we can create a Notes form to display the information:



Title:	Title
Authors:	Authors
Publisher:	Publisher
ISBN:	ISBN
ASIN:	ASIN
State:	State
Notes Version:	NotesVer
Categories:	Category

The first five lines—title, authors, publisher, ISBN, and ASIN—contain just the information that is stored from the BookTracker interface. The State field lets us tag this book as no longer published or not relevant (for instance, getting a hit on a book that is not related to Notes/Domino). The Notes Version field lets us identify which versions of Notes the book applies to. The Category field lets us identify which categories the book belongs in—Administration, Programming, and so on.

You also need two Notes views. One view is called (LUBookASIN). The first (and only) sorted column contains the ASIN field. The ASIN is Amazon's unique ID for each of their products. The ISBN can be null for cases like electronic books. This hidden lookup view is used for updating the status of books, so we know when a book was last on Amazon's Web site; if a book is no longer found by a keyword search, it is no longer published because Amazon doesn't return unpublished books in its search engine. If a book is found in a keyword search, but is not found by looking up the ASIN in our database, it is a new book and has to be added.

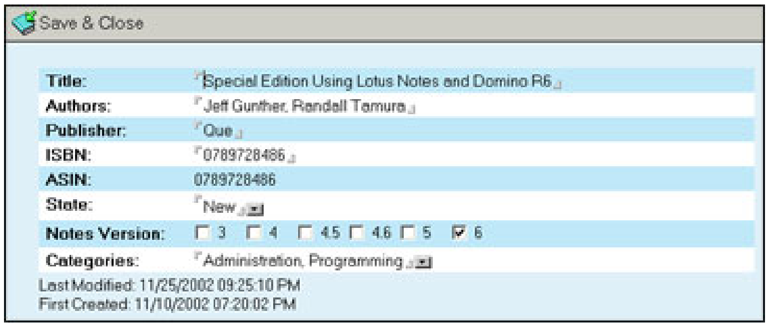
The second view is called Books\by Status. The first categorized column in this view is the Status field. The second column is the Notes Version field. The third column is the Title field. You can use this view to manage books that were added by our AmazonUpdate program.

Saving new books to the Notes database
Now that we have a place to put each of the books that our Java application has found, we can write a BookTracker class that understands how to save to our Notes database. It's slightly more complicated than the dump debug class, but not much more so. It uses the Notes Java API, so you need the notes.jar file (in your Notes directory with notes.exe) in your classpath:

```
import lotus.domino.*;

public class NotesBookTracker implements BookTracker {
	lotus.domino.Session s;
	Database db;
	View booksview;

	// constructor
	public NotesBookTracker(String server, String dbname) throws NotesException {
		// initialize Notes access
		NotesThread.sinitThread();
		s = NotesFactory.createSession();

		// get book ISBN view
		db = s.getDatabase(server, dbname);
		booksview = db.getView("(LUBookASIN)");
	}

	// Java pseudo-destructor
	public void finalize() throws NotesException {
	// clean up Notes
	NotesThread.stermThread();
	}

	// update book from amazon query
	public void updateBook(
		String title,
		String ASIN,
		String ISBN,
		String authors,
		String publisher)
		throws NotesException {
		// check for blank ISBN
		if (ISBN == null) {
		System.out.println(title + " has a null ISBN!");
	}
		// check for blank ASIN
		if (ASIN == null) {
		System.out.println(title + " has a null ASIN!");
	}

	Document doc;
	// see if we can find the book
	doc = booksview.getDocumentByKey(ASIN, true);
	if (doc == null) {
	// new book, so we have to add it to the database
	doc = db.createDocument();
	doc.replaceItemValue("Form", "Book");
```

```
    doc.replaceItemValue("Title", title);
    doc.replaceItemValue("ASIN", ASIN);
    if (ISBN != null) { // E-books don't have an ISBN
    doc.replaceItemValue("ISBN", ISBN);
    }
    if (authors != null) {
    doc.replaceItemValue("Authors", authors);
    }
    if (publisher != null) {
    doc.replaceItemValue("Publisher", publisher);
    }
    doc.replaceItemValue("State", "Added");
    }// tag it w/ timestamp so we know when it was last found
    // so we can expire unfound books in a scheduled agent
    DateTime timenow = s.createDateTime("Today");
    timenow.setNow();
    doc.replaceItemValue("LastFound", timenow);
    // save document
    doc.save();
    }
}
```

Editing the forms and viewing the books
After you receive the information from Amazon.com, edit the State, Notes Version, and Categories fields in the Notes database. The following screen shows a document in edit mode.

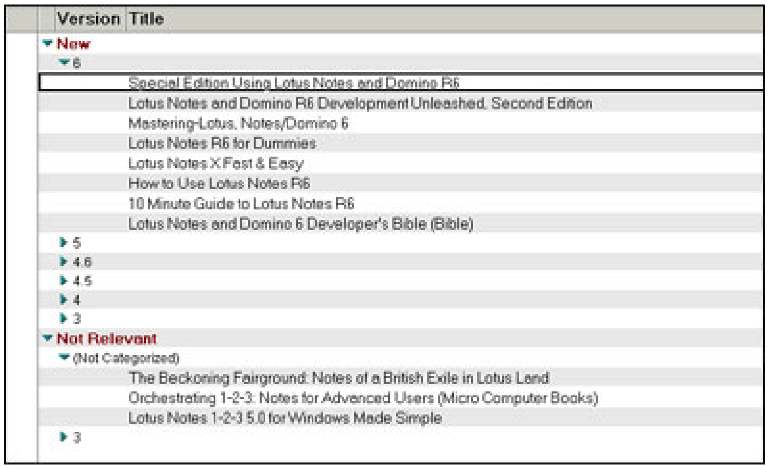


Save & Close

Title:	Special Edition Using Lotus Notes and Domino R6
Authors:	Jeff Gunther, Randall Tamura
Publisher:	Que
ISBN:	0789728486
ASIN:	0789728486
State:	New
Notes Version:	3 4 4.5 4.6 5 [✓ 6]
Categories:	Administration, Programming

Last Modified: 11/25/2002 09:25:10 PM
First Created: 11/10/2002 07:20:02 PM

After editing the fields of the new books that the Java agent found, you should have a useful view that looks like this:

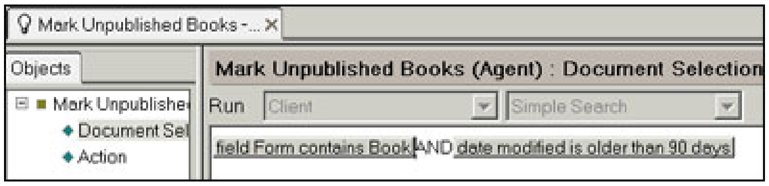


```
Version  Title
▼ New
  ▼ 6
           Special Edition Using Lotus Notes and Domino R6
           Lotus Notes and Domino R6 Development Unleashed, Second Edition
           Mastering Lotus, Notes/Domino 6
           Lotus Notes R6 for Dummies
           Lotus Notes X Fast & Easy
           How to Use Lotus Notes R6
           10 Minute Guide to Lotus Notes R6
           Lotus Notes and Domino 6 Developer's Bible (Bible)
  ▶ 5
  ▶ 4.6
  ▶ 4.5
  ▶ 4
  ▶ 3
▼ Not Relevant
  ▼ (Not Categorized)
           The Beckoning Fairground: Notes of a British Exile in Lotus Land
           Orchestrating 1-2-3: Notes for Advanced Users (Micro Computer Books)
           Lotus Notes 1-2-3 5.0 for Windows Made Simple
  ▶ 3
```

Marking books as no longer published

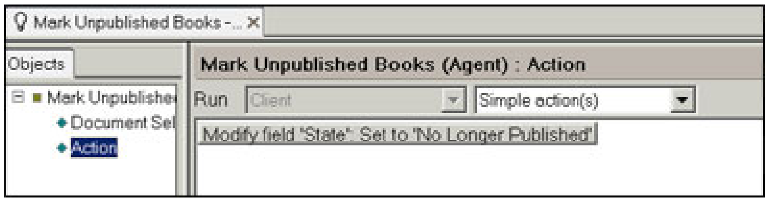
This is something our BetterEyes application does a much better job with than Amazon's Eyes service. The Eyes service never notified you when books were no longer published. Amazon had another feature for Amazon Associates that let you know when Web site visitors referenced your links to products that were no longer available; you would then have to edit these links by hand.

In Notes, we can create a scheduled agent to mark these books as no longer published. You'll note that whenever our AmazonUpdate application runs, it updates the documents that already exist and stores a LastFound date. We can then create a scheduled agent that runs every 30 days with the following document selection:


Mark Unpublished Books -...
Objects
Mark Unpublishe
Document Sel
Action
Mark Unpublished Books (Agent) : Document Selection
Run
Client
Simple Search

field Form contains Book AND date modified is older than 90 days

and this simple action:


Mark Unpublished Books -...
Objects
Mark Unpublishe
Document Sel
Action
Mark Unpublished Books (Agent) : Action
Run
Client
Simple action(s)

Modify field "State": Set to 'No Longer Published'

That's all there is to it. We can now create a view that lists only books that are available for purchase by checking that the State field is not No Longer Published.

AmazonUpdate as a Notes/Domino 6 Java agent

Converting the AmazonUpdate Java application to a Notes/Domino 6 Java agent is relatively straightforward. You can't do this using R5 because R5 has an older Java Runtime 1.1.8, whereas Notes/Domino 6 has JRE 1.3.1. The first thing you have to do is add the additional JAR files to the Notes and Domino classpath.

Handling the Java classpath is different in Notes/Domino 6 from R5 because Notes/Domino 6 has a more updated JVM. In your Notes or Domino directory, there is a JVM subdirectory where the JVM binaries and JAR files are kept. If you want to add JAR files to your agent's classpath, you can add them to the jvm\lib\ext directory. You should add axis.jar, jaxrpc.jar, saaj.jar, commons-logging.jar, commons-discovery.jar to this directory (check the Axis documentation to see what is required because they may change the JAR files they distribute).

The AmazonUpdate.jar file should be added to the jvm\lib\ext directory as well. This JAR file includes all the CLASS files that we created for our application as well as the SOAP proxy classes. You create this JAR file by going to the directory you put your application files in (c:\AmazonUpdate is assumed in the path above) and typing:

```
jar -cvf AmazonUpdate.jar *.class com\amazon\soap\axis\*.class
cd build\axis
jar -cvf ..\..\AmazonUpdate.jar com\amazon\soap\axis\*.class
```

The reason we do this, rather than import all our JAVA files into the IDE, is that Domino Designer slows down significantly if you try to import a lot of JAVA files into it as it tries to decide which pieces of code to display expanded or collapsed. It's a lot faster to just put your code that doesn't change into a JAR file and place it in the classpath.

In addition, you need to add a SAX API compatible XML parser to the path. The simplest one to add is Apache's Crimson parser, which comes in the form of a file named crimson.jar. Unfortunately, you can't add it to the jvm\lib\ext directory because Notes/Domino 6 has its own DOM parser in the xml4j.jar file in the Notes executable directory. This has an old version of the javax.xml.Node class that is missing a method that is referenced by Axis. Luckily, we can put our classes in front of the search path that includes the xml4j.jar file by using the JavaUserClasses Notes.ini variable. Add this line of code to your Notes.ini file (assuming the file is in the C:\JARs directory):

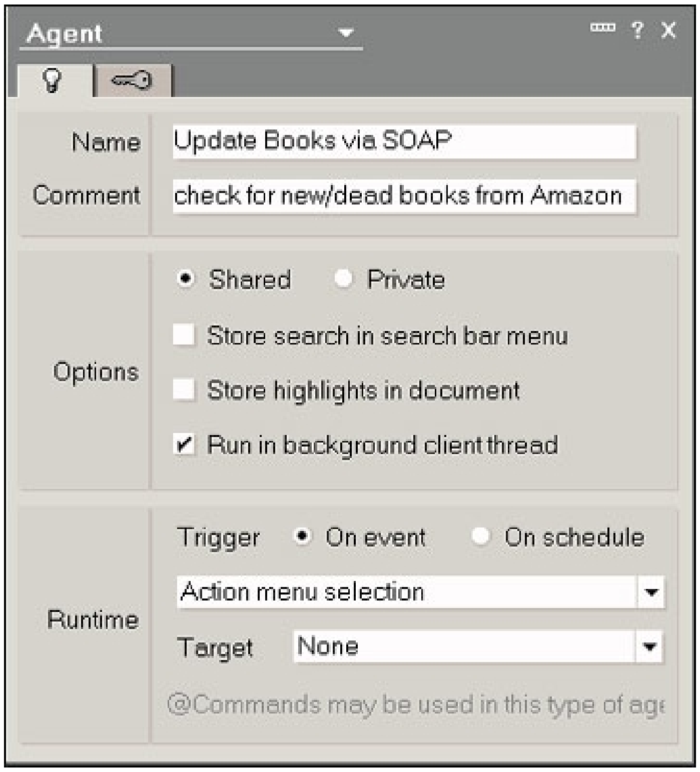
```
JavaUserClasses=c:\JARs\crimson.jar
```

Be careful with how many paths you put into the JavaUserClasses variable. As is true of all Notes.ini variables, each one has a limit of 255 characters. It's easy to run over this limit because of all the useful Java JAR files that are available. The only workaround is to put all of them in a one-character directory name at the root of a drive, but this workaround doesn't last long if you keep adding JAR files.

After you complete this setup work, you can create a Java agent in Domino Designer. All you need is a few lines of code:

```
import lotus.domino.*;

public class JavaAgent extends AgentBase {

    public void NotesMail() {

        try {
            Session session = getSession();
            AgentContext agentContext = session.getAgentContext();

            BookTracker tracker = new DumpBookTracker();
        //BookTracker tracker = new NotesBookTracker("SlipGate","Notes")
            AmazonUpdate.getAmazonBooks("lotus domino", tracker);
            AmazonUpdate.getAmazonBooks("lotus notes", tracker);

        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Again, we're using the DumpBookTracker API to test with. Set the agent to run manually from the menu, so you can test this in your Notes client:



Name	Update Books via SOAP
Comment	check for new/dead books from Amazon
Options	[● Shared] ○ Private
	☐ Store search in search bar menu
	☐ Store highlights in document
	☑ Run in background client thread
Runtime	Trigger [● On event] ○ On schedule
	Action menu selection
	Target: None
	@Commands may be used in this type of age

Before you run the agent, in the Notes client, choose File - Tools - ShowJavaConsole, so you can see all the output from the System.out.println calls in the BookTracker class. Error messages, including errors about not being able to find some classes, from the Java Virtual Machine are also displayed in this window.

Conclusion

We've created a very useful Java application that uses Amazon Web Services. The open source Axis SOAP proxy generator makes this task almost too easy because it seems like we're just using regular Java classes; we don't have to worry about how to reach the Web service or how to talk to it as long as the Web service publishes a WSDL file.

By storing the information from Amazon Web Services in a Notes database, we can build an application that is more useful than Amazon's former Eyes service because we can track books that are no longer published as well as new books that were added. What's more, the Notes database can be used over the Web by putting it on a Domino server. All of the Notes work was done in a few hours and provides a very useful interface to the information that Amazon kindly supplied via Amazon Web Services.

In our next article, we'll rebuild the same application using DB2 and JavaServer Pages (JSPs). Then, we'll rebuild it using EJBs with a few automation tools (MiddleGen and XDoclet and Ant) that make EJBs much less painful than they used to be.

ABOUT THE AUTHOR
Ken Yee has been a consultant and Lotus Business Partner since the inception of the program. He has done software development for 14 years and is always looking for interesting J2EE/Domino integration projects. His company, **KEY Enterprise Solutions**, has done Notes, Domino, IIS/ASP, Java, ActiveX/COM, and C++ development and administration projects for **Lotus**, **Inso/Stellent**, **Logica**, **eVelocity**, **World Bank**, and **Analysis Group**. KEY Enterprise Solutions maintains the **Notes/Domino FAQ** (the first Notes FAQ on the net) as a service to the Notes community and the **Java Servlet FAQ** for the Java community.