Level: Intermediate
Works with: Designer 5.0
Updated: 02-Jan-2001

by
Kyla
Town

The development of applications using XML (Extensible Markup Language) is rapidly picking up steam. XML provides the power to create markup languages that describe data. You can use it to share data across a network, and even across applications.

Where does Domino Designer fit into this new technology? In addition to workflow and security capabilities, Domino Designer provides a medium for writing XML and then serving the XML data to a parser. Domino Designer is a powerful development environment that provides the layers of security needed to protect data, including database access control and field encryption.

To illustrate how to integrate XML into a Domino application, this article will walk you through the process of creating elements of an online auto parts catalog for a fictional store called Acme Auto Parts. You will learn how to:

- Create forms and pages that use XML
- Create a view that displays XML data
- Embed the view into a page
- Apply an XSL stylesheet to XML data programmatically
- Use agents to access XML from forms and views
- Use Java to export and generate XML from a Notes database
- Use agents to access data in other Domino databases

You can also download two sample databases, **Acme.nsf and Zippy.nsf**, from the Iris Sandbox on Notes.net. These databases contain all the design elements and code discussed in this article.

Please note the following requirements:

- The sample databases require the Notes R5.0.3 client and Microsoft Internet Explorer 5.5. Specifically, the information in this article and in the sample databases works with Internet Explorer 5.50.4522.1800. Other versions of Internet Explorer may not perform as described in this article.
- You need Domino Designer R5.0.3.
- The Java agent described in this article requires the IBM XML4j parser and the LotusXSL

processor. These tools are included with Domino Designer R5.0.3 or later. You can also download the IBM XML4j parser from the **IBM Alphaworks** Web site. You can download the LotusXSL processor from the **Lotus Developer Network** Web site.
- This article assumes you already have a basic working knowledge of XML and Domino Designer R5.0.3.

For general information about creating databases, outlines, agents, and framesets, see **Domino 5 Designer Help**. For general information about XML and XSL (Extensible Stylesheet Language), see the **World Wide Web Consortium (W3C)**.

The scenario

In this example, Acme maintains an e-commerce site that uses a Domino database to manage an online catalog of auto parts and tools. Acme carries parts from more than one vendor. Acme chose to use XML for this application because with XML as the common language to describe part information, a purchasing agent at Acme can pull information from various vendors about part pricing and availability directly into the Acme Auto Parts Catalog. Customers can access this application for up-to-date information about parts and tools from more than one distributor.

Creating a form that uses XML

You can use a form to enter XML tags and include fields within the tags for data. The result is an XML document with data that is meaningful when delivered to an XML parser. When you use XML elements on a form or page, you must follow the rules for constructing well-formed XML and you must properly format the XML tags. Additionally, the XML in a form or page should be accessed programmatically when using Microsoft Internet Explorer 5.5. Accessing pages or forms with XML in them directly will cause the browser to hang.

This step describes how to create the Parts form in the Acme Auto Parts Catalog.

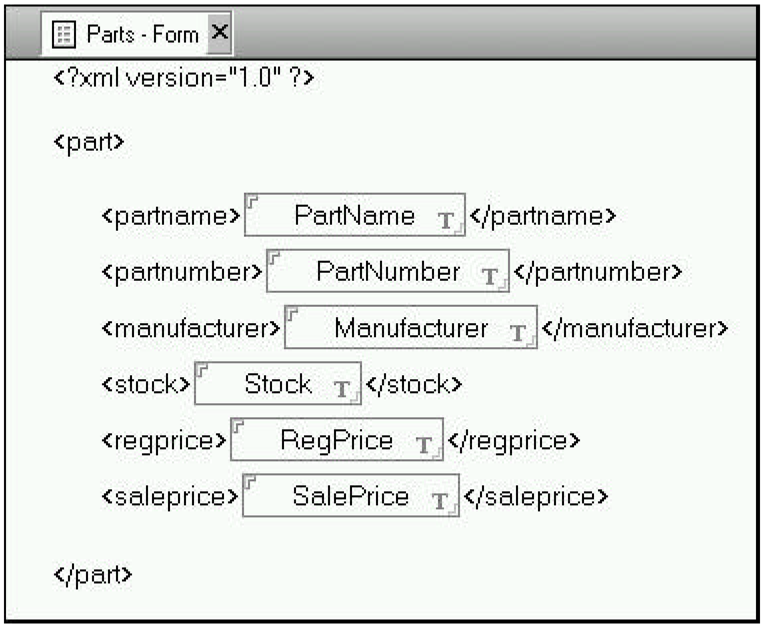
1. Create a blank database called XML Exercise and open it in Designer.
2. Choose Forms in the Design pane for XML Exercise.
3. Click the New Form button.
4. In the Form properties, type Parts as the form name.
5. Type the following XML declaration as the first line in the form.

   ```
   <?xml version="1.0" ?>
   ```
6. Type the following XML tags beginning on the line below the XML declaration.

   ```
   <part>

   <partname></partname>

   <partnumber></partnumber>

   <manufacturer></manufacturer>

   <stock></stock>

   <regprice></regprice>

   <saleprice></saleprice>

   </part>
   ```

   Notice that all of the tags are contained by the <part></part> tags. These are called parent tags, and they describe the beginning and end of a chunk of data. In this case, they describe the beginning and end of the data describing one part. A tag contained within the parent tag is called the child tag.

7. Create a field between each child element tag to hold the data you want to mark with XML. Here is what the form should look like when you're done:



```
<?xml version="1.0" ?>

<part>

    <partname> PartName </partname>
    <partnumber> PartNumber </partnumber>
    <manufacturer> Manufacturer </manufacturer>
    <stock> Stock </stock>
    <regprice> RegPrice </regprice>
    <saleprice> SalePrice </saleprice>

</part>
```

8. Choose Design - Form Properties.
9. On the Advanced tab, click "Treat document contents as HTML." This property tells Domino to pass all of the document text to the HTTP requestor without generating HTML tags.
10. Save and close the form.
11. Choose Create - Parts and add some data to the form.

Creating a view to display XML data

Now that you have a form to work with, you will need to create a view to display the XML documents. Views enable you to select which documents are rendered in XML and to track information. Views are also useful because they can deliver several documents at a time while a form can only deliver one document at a time. This step describes how to create a Parts view like the one in the Acme database.

1. Create a view and open it.
2. Choose Edit - Properties to open the View properties box.
3. Click the View Info tab and type Parts in the Name field.
4. On the Advanced tab, select "Treat view contents as HTML." Domino generates HTML for the contents of a view if this property is not selected. The contents of a view that is embedded into a page are not visible in the browser if this property is not selected.
5. On the Objects tab, click "View Selection" and type the following formula to include the Parts form in the view.

   SELECT Form = "Parts"
6. Add one column to the view for each child element in the form.
7. Click the first column of the view.
8. Type a column formula using the following syntax:

   "<PARENT><CHILD>"+*fieldname*+"</CHILD>"

   For example, the formula for the first column should look like this:

```
"<part><partname>"+PartName+"</partname>"
```

9. Click the second column and type a column formula into the Script area using the following syntax:

   "<CHILD>"+*fieldname*+"</CHILD>"

   For example, the formula for the second column should look like this:

   ```
   "<partnumber>"+PartNumber+"</partnumber>"
   ```
10. Repeat the previous step for each XML element except the last one.
11. For the last child element, use the following syntax:

    "<LASTCHILD>"+*fieldname*+"</LASTCHILD></PARENT>"

    For example, the formula for the last column should look like this:

    ```
    "<saleprice>"+SalePrice+"</saleprice></part>"
    ```

Tip: If you have more than one element for a column, you can add a semicolon (;) at the end of the first column formula, and then add the column formula for the next element below it, as follows:

"<PARENT><CHILD>"+*fieldname*+"<\CHILD>";
"<CHILD>"+*fieldname*+"<\CHILD>";
"<CHILD>"+*fieldname*+"<\CHILD>"

Embedding a view in a page

Embedding a view in a page allows you to add the root element to your XML. Remember that the XML in a form or page should be accessed programmatically when using Microsoft Internet Explorer 5.5. Accessing pages or forms with XML in them directly will cause the browser to hang.

To embed the view in a page:

1. Create a new page.
2. Choose Design - Page Properties.
3. Type GetParts in the Name field.
4. Click the Page Info tab and select "Web Access: Treat page contents as HTML."
5. Type an XML declaration and an open root element tag above the place where you want the embedded view to display. For example:

   ```
   <?xml version="1.0" ?>
   <partscatalog>
   ```
6. Place the cursor where you want the embedded view to display.
7. Choose Create - Embedded Element - View, and then choose Parts.
8. Click the embedded view and select Element - View Properties.
9. On the Info tab:
   • Type Main in the Target Frame field for the double click option.
   • Select "Web Access: Display Using HTML."
10. Type a close root tag on the page below the view. For example:

    ```
    </partscatalog>
    ```

Accessing XML with LotusScript

Now that you have created a form, view, and page, you can access the XML you have created programmatically with LotusScript, JavaScript, or Java. Let's look at how to access XML using a server-side LotusScript agent. First, here's the agent:

```
Dim source As Variant

‘The sourceFile variable can be a URL, a file on the server, or a Notes
'document. If you want to access a specific document, you can use the
'OpenDocument URL command and the UNID of the document you want
'to access. For example, http://localhost/Acme.nsf/unid/OpenDocument
'where unid is the UNID of the document.
'sourceFile="http://localhost/Acme.nsf/GetParts?OpenPage"

‘This line creates the Microsoft.XMLDOM object.
Set source = CreateObject("Microsoft.XMLDOM")
source.async = False

‘This line loads the XML into the parser.
source.load(sourceFile)

'This line prevents Domino from sending default headers.
Print "Content-type: text/xml"

'This line displays the resulting XML in the browser.
Print source.xml
```

In order to make this agent accessible from a browser, you can add an action entry to the database outline. To see an example of this, be sure to download the **Acme.nsf and Zippy.nsf** databases from the Iris Sandbox on Notes.net. The Acme database runs this agent from the Parts (server-side LS) outline entry.

Here's how to add the agent to the Acme database outline:

1. Open the database outline.
2. Click New Entry.
3. Type a label for the entry.
4. Set the Content Type to Action.
5. Click the @ button.
6. Type the following formula:

   ```
   @Command([ToolsRunMacro];"Generate XML")
   ```

Applying an XSL stylesheet to form data

Up to this point, you've seen that XML only describes data and doesn't define its appearance. The presentation is not important for computer-to-computer transactions, but if you're presenting data to a person, a stylesheet can help to make it more readable. A stylesheet can also determine which parts of the data are displayed and whether or not the data is converted to HTML, or to another form of XML. Some browsers provide simple default styles for popular elements such as <Para>, <List>, and <Item>, but generally you must use a stylesheet to describe the data format.

There are two types of stylesheets you can use with XML:

- An extensible stylesheet language (XSL) to describe how to transform XML into HTML or into another version of XML
- A cascading stylesheet (CSS) to style XML directly on Web browsers that support CSS

The Acme Auto Parts Catalog uses XSL to translate XML documents into HTML. The stylesheet, acmepartscatalog.xsl, appears below. Comment lines (indicated by <!-- and -->) explain the styles used in the stylesheet.

Note: The XSL standard used by IE 5.5 is different from the standard recommended by the W3C. Stylesheets created to work in earlier versions of IE 5.5 may not work with future versions of the parser and XSL processor.

```
<?xml version="1.0"?>

<!--This line is for IE only-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<!--root node-->
<xsl:template match="/" >

<!--This line sets the table border to 3 pixels, inset, black-->
<TABLE STYLE="border:3px inset black">

<!--The first row of the table. The font of the text in this row is set 10 point, boldface, Verdana, and the background
color of the row to light grey.-->
<TR STYLE="font-size:10pt; background-color: lightgrey; font-family:Verdana; font-weight:bold">

<!--This is the text that will appear in the first cell of the first row of the table. All of the cells in this row will have the
attributes defined in the line above.-->
<TD>Part Name</TD>

<!--This is the text that will appear in the second cell of the first row of the table-->
<TD>Part Number</TD>

<!--This is the text that will appear in the third cell of the first row of the table.-->
<TD>Manufacturer</TD>

<!--This is the text that will appear in the fourth cell of the first row of the table.-->
<TD>Stock</TD>

<!--This is the text that will appear in the fifth cell of the first row of the table.-->
<TD>Regular Price</TD>

<!--This is the text that will appear in the sixth cell of the first row of the table.-->
<TD>Sale Price</TD>

<!--The end of the first row in the table.-->
</TR>

<!--This line applies the defined style to the data in each child element contained by the root element
<partscatalog> and the parent element <part>, then puts it in the table in order by part name. Note that order-by is
not the field name, but the XML tag that is associated with the field.-->
<xsl:for-each select="partscatalog/part" order-by="partname">

<!--The second row and subsequent rows of the table. The font is set to Verdana, 10 point. The cell padding is set
to 0 pixels and 6 pixels.-->
<TR STYLE="font-family:Verdana; font-size:10pt; padding:0px 6px">

<!--The data contained by the <partname> tag will be placed in the first column.-->
```

```
<TD><xsl:value-of select="partname" /></TD>

<!--The data contained by the <partnumber> tag will be placed in the second column.-->
<TD><xsl:value-of select="partnumber" /></TD>

<!--The data contained by the <manufacturer> tag will be placed in the third column.-->
<TD><xsl:value-of select="manufacturer" /></TD>

<!--The data contained by the <stock> tag will be placed in the fourth column.-->
<TD><xsl:value-of select="stock" /></TD>

<!--This line puts the data contained by the <regprice> tag in the fifth column. A dollar sign ($) will appear in each
cell in the
fifth column before the price data.-->
<TD>$<xsl:value-of select="regprice" /></TD>

<!--The data contained by the sale price tag will be placed in the sixth column. A dollar sign ($) will appear in each
cell in the sixth column before the price data. The background color of the cell is set to light grey.-->
<TD STYLE="background-color:lightgrey">$
<xsl:value-of select="saleprice" /></TD>

<!--This line ends each row of the table. Note that a third row, fourth row, and so on will be created until all of the
data has been rendered.-->
</TR>

</xsl:for-each>

</TABLE>

</xsl:template>

</xsl:stylesheet>
```

To use an XSL stylesheet in a Domino application:

1. Create a new page.
2. Choose Design - Page Properties.
3. Enter a name with the extension XSL in the Name field. For example: acmepartscatalog.xsl.
4. Select "Web Access: Treat page contents as HTML."
5. Write your own stylesheet, or copy the acmeautopartscatalog.xsl code above into the XML Example database.
6. Save the page with an XSL extension.
7. Create a LotusScript or JavaScript agent to access the XML and apply the stylesheet programmatically.

Accessing XML with JavaScript

The JavaScriptXML page in the Acme.nsf database uses client-side JavaScript to access the XML in the GetParts page and apply the acmepartscatalog.xsl stylesheet to it.

To access XML with client-side JavaScript:

1. Create a new page.
2. Choose Design - Page Properties.

3. Type JavaScriptXML in the Name field.
4. Click the Page Info tab and verify that "Web Access: Treat page contents as HTML" is not selected.
5. Type the following division tags on the page where you want your data to display:

   ```
   <div id="HTMLresults"></div>
   ```

   Note: HTMLresults is an attribute that is used in the JSHeader object described below.
6. Highlight the <div id="HTMLresults"></div> tags and click Text -- Pass-Thru HTML. The tags will appear with a grey background indicating that they are marked as pass-thru HTML.
7. Optionally, add HTML Head Content.
8. Click JSHeader and type or paste the following JavaScript into the Script area:

```
var HTMLresults;
var source;
var style;

//Defines loadXML() as a function.
function loadXML() {

        //Creates a new ActiveXObject called source which is the
        //Microsoft.XMLDOM parser.
        source = new ActiveXObject("Microsoft.XMLDOM");

        //Creates a new ActiveXObject called style which is the
        //Microsoft.XMLDOM parser that is installed with IE 5.5.
        //This object will be used to apply the stylesheet to the XML.
        style = new ActiveXObject("Microsoft.XMLDOM");

        source.async = false;
        style.async = false;

        //Sets the validateOnParse property to false.
        source.validateOnParse = false;

        //Loads the GetParts page into the parser using
        //the openPage URL command.
        Source.load("http://localhost/Acme.nsf/GetParts?openPage");

        //Loads the acmepartscatalog.xsl stylesheet into the
        //parser using the openPage URL command.
        Style.load("http://localhost/Acme.nsf/ acmepartscatalog.xsl?OpenPage");

                //Error handler. If there is an error, call the
                //showError() function.
                if(source.parseError.errorCode != 0) {
                showError();
        }
        //If there is no error, call the doTransform() function.
        DoTransform();
}

//Defines doTransform() as a function.
function doTransform() {
        //If the getReadyState() function returns true,
        //then transform the node using the
        //stylesheet and put the results inside the tags with
```

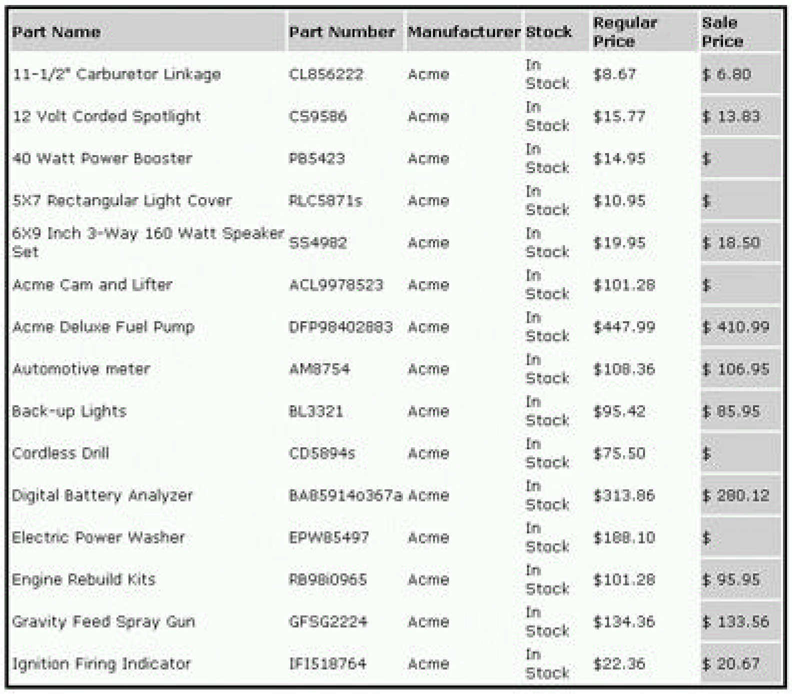
```
//the HTMLresults attribute.
        if (getReadyState()){
                resulting = source.transformNode(style);
                document.all.item("HTMLresults").innerHTML = resulting;
        }
        //If the getReadyState() function doesn't return
        //true, then refresh.
        Self.refresh;
}

//Defines the getReadyState function.
function getReadyState() {
        alert("ready state: " + source.readyState);
        if (source.readyState == 4) {
        return true;
        }
        setTimeout("getReadyState()", 100);
}

//Defines the showError() function.
function showError() {
        var strError = new String;
        var err = source.parseError;
        strError = 'Error!\n' +
        'file url: '+err.url +' \n'+
        'line no.:'+err.line +'\n'+
        'char: '+ err.linepos + '\n' +
        'source: '+err.srcText+'\n'+
        'code: '+err.errorCode+'\n'+
        'description: '+err.reason+'\n';
        document.all.item("HTMLresults").innerHTML = strError;
}
```

9. In the OnLoad event, type or paste the following JavaScript:

```
//Calls the loadXML() function when the page is
//loaded into the browser.
loadXML();
```

10. Save the page.

11. Choose Design - Preview In Web Browser - Internet Explorer.
Note: You are able to preview this page directly in Internet Explorer 5.5 because the XML in the GetParts page is being accessed programmatically via JavaScript.



Part Name	Part Number	Manufacturer	Stock	Regular Price	Sale Price
11-1/2" Carburetor Linkage	CL856222	Acme	In Stock	$8.67	$ 6.80
12 Volt Corded Spotlight	CS9586	Acme	In Stock	$15.77	$ 13.83
40 Watt Power Booster	PB5423	Acme	In Stock	$14.95	$
5X7 Rectangular Light Cover	RLC5871s	Acme	In Stock	$10.95	$
6X9 Inch 3-Way 160 Watt Speaker Set	SS4982	Acme	In Stock	$19.95	$ 18.50
Acme Cam and Lifter	ACL9978523	Acme	In Stock	$101.28	$
Acme Deluxe Fuel Pump	DFP98402883	Acme	In Stock	$447.99	$ 410.99
Automotive meter	AM8754	Acme	In Stock	$108.36	$ 106.95
Back-up Lights	BL3321	Acme	In Stock	$95.42	$ 85.95
Cordless Drill	CD5894s	Acme	In Stock	$75.50	$
Digital Battery Analyzer	BA85914o367a	Acme	In Stock	$313.86	$ 280.12
Electric Power Washer	EPW85497	Acme	In Stock	$188.10	$
Engine Rebuild Kits	RB98i0965	Acme	In Stock	$101.28	$ 95.95
Gravity Feed Spray Gun	GFSG2224	Acme	In Stock	$134.36	$ 133.56
Ignition Firing Indicator	IFI518764	Acme	In Stock	$22.36	$ 20.67

Exporting and generating XML using a Java agent

You can use Java to export and generate XML from a Notes document as well. For this example, you will need the IBM XML4J parser and the LotusXSL processor.

To install the IBM XML4j parser and the LotusXSL processor:

1. Download the XML4j parser from the **IBM Alphaworks** Web site.
2. Download the LotusXSL processor from the **Lotus Developer Network** Web site.
3. Add the XML4j.jar and LotusXSL.jar files to your Path variable.

The Acme.nsf database uses the JavaAgent agent to get the XML data in the Parts page.

To export data and generate XML using a Java agent:

1. Create an agent.
2. Select Java from the Run menu in the agent.
3. Write some Java code to process the document and generate XML using the XML4j parser. See the agent named, JavaAgent in the **Acme.nsf** database for an example.

To access the Java agent from the browser:

1. Open the database outline.
2. Click New Entry.
3. Type a label for the entry.
4. Set the Content Type to URL.
5. Click the @ button.
6. Type the following URL command:

```
./agentName?OpenAgent&db=Acme.nsf&unid=documentUNID
```

Where documentUNID is the universal ID of the document you want to export and render in XML and agentName is the name of the Java agent you have just created.

Accessing information in another database

In this example, Acme wants to include information about parts that are supplied by Zippy Auto, an affiliate of Acme. Zippy maintains its own regional catalog with basically the same design as Acme's, but it's in a Domino database with no XML tags. You can write an agent that will retrieve the information from Zippy's database and display it for the client.

To create a LotusScript agent to access information in another database:

1. Create an agent.
2. Select LotusScript from the Run menu in the agent.
3. Copy and paste the LotusScript shown below into the Script area, or write your own.

The following LotusScript code demonstrates how to get the information and transform it into XML. Note that the print statement will print the information to the status bar if the agent is run in the Notes client. If the agent is run in the browser, it will print to a page in the frameset.

```
Dim session As New NotesSession
'db is the current database
Dim db As NotesDatabase
'dbZippy is the database you want to access
Dim dbZippy As New NotesDatabase("","zippy.nsf")
Dim doc As NotesDocument
Dim view As NotesView

Set db = session.CurrentDatabase
'gets the Notes Documents view in Zippy's database
Set view = dbZippy.GetView( "Notes Documents" )
'gets the first document in the Notes Documents view
Set doc = view.GetFirstDocument

'prevents Domino from sending default headers
Print "Content-type: text/xml"
'this is the XML declaration
Print "<?xml version='1.0' encoding='UTF-8'?>"
'this is the XSL processing instruction
Print "<?xml:stylesheet type='text/xsl' href='acmepartscatalog.xsl' ?>"
'root element
Print "<partscatalog>"

'loop as long as there are document objects available
While Not ( doc Is Nothing )

'send the parent element for each parts document
Print "<part>"

          'send appropriate markup, pulling field content
          'from the document object
          Print "<partname>"+doc.PartName(0)+"</partname>"
          Print "<partnumber>"+doc.PartNumber(0)+"</partnumber>"
```

```
        Print "<manufacturer>"+doc.Manufacturer(0)+"</manufacturer>"
        Print "<stock>"+doc.Stock(0)+"</stock>"
        Print "<regprice>"+doc.RegPrice(0)+"</regprice>"
        Print "<saleprice>"+doc.SalePrice(0)+"</saleprice>"
        'close the part element tag
        Print "</part>"
        'get the next document in the view
        Set doc = view.GetNextDocument( doc )


Wend
'close the root element tag
Print "</partscatalog>"
```

Make this agent accessible from the browser by adding an action entry to the database outline. The Acme database runs this agent from the Zippy Tools outline entry.

To add the action to the Acme database outline:

1. Open the database outline.
2. Click New Entry.
3. Type a label for the entry.
4. Set the Content Type to Action.
5. Click the @ button.
6. Type the following formula:

   ```
   @Command([ToolsRunMacro];"Generate XML")
   ```

Conclusion

Domino Designer provides an opportunity to integrate XML into Domino applications that share data across a network in a secure manner. You can add XML tags to forms and use views to determine which documents are rendered in XML. You can use stylesheets to create the look you want for your data. Agents are used to access XML in pages and forms, generate XML from non-XML documents, and retrieve data from other Domino applications. Hopefully, this article has provided you with some ideas about using XML in your own applications.

ABOUT THE AUTHOR
Prior her current position as a technical writer at IBM, Kyla was a member of the Domino Designer documentation team at Lotus for two years. While at Lotus, she worked on the *Application Development with Domino Designer* and the *Domino Designer Programming Guide* manuals. When she isn't writing, Kyla enjoys knitting, crocheting, and watching one of her four cats chase the yarn as she works.