XML training wheels

An XSLT and Java-based tool for producing tutorials -- custom-built for developerWorks but ready to adapt to your own use

Doug Tidwell (dtidwell@us.ibm.com)
Cyber evangelist, developerWorks
June 2001

See how developerWorks produced a custom XSLT application with Java-based open-source tools that automates the tedious work of producing the developerWorks HTML-based tutorials. Known as the Toot-O-Matic, the tool now is available for any developer either to inspect as an XSLT exemplar or to tailor to your own training needs. Doug Tidwell explains the design goals and the XML document design. He also describes how the 13 code samples demonstrate the techniques used in generating a truckload of HTML panels full of custom graphics, a ZIP file, and two PDF files from a single XML source document.

Contents:

Here at developerWorks, we're pleased to release the source of the Toot-O-Matic, the XML-based tool we use to create our tutorials. In this article, we'll discuss the design decisions we made when we built the tool, talk about how you can use it to write your very own tutorials, and talk a little bit about how the source code is structured. We hope you find the tool useful, and that it will give you some ideas about how to use XML and XSLT style sheets to manipulate structured data in a variety of useful ways.

Toot-O-Matic design goals

The Toot-O-Matic project started with several design goals:

- Make it easier to create developerWorks tutorials
- "Eat our own dog food" (show that we actually use the technologies we advocate)
- See how much we could accomplish through XSLT style sheets

First I'll elaborate on these goals, then I'll talk about the actual design of the tutorials.

Make it easier to create tutorials

When we built our first tutorials at developerWorks a couple of years ago, they were incredibly tedious to create. Authors and editors would write and edit the content in a tool such as Microsoft Word, then we would start the publishing process. Our first step was typically to create a PDF version of the tutorial. High-quality printable versions of our tutorials are popular, and it's pretty easy to create them from a formatted document in Microsoft Word (much easier than from a slew of HTML files).

Once that word-processing document was done, we converted the tutorial into HTML. We then took that single HTML file, broke it into small pieces, and added the standard IBM header and footer to each small piece. This gave us a number of HTML files (usually 50 to 100) that we had to link together. We'd link them so that if you're looking at the third panel in a section, clicking **Next** should take you to the fourth panel, and clicking **Previous** should take you to the second panel. We also then needed to create a menu panel; from the menu panel, you can link directly to the first panel of any particular section. Finally, each panel had mouseover effects that had to be tested.

While the writer and editor worked on the actual content, our graphic designers created artwork for the titles of the sections and the panels of the tutorial itself. To ensure a consistent look for the heading text, our designers created graphics that contained that text, drawn on the appropriate background. For some titles, the designers made both plain and highlighted versions for the mouseover effects.

As you can probably imagine, a great deal of the tutorial-building process was hand coded and error prone (particularly when yours truly was feverishly finishing a tutorial at 5:30 in the morning so it would be on the site by sunup). We wanted to automate as many of these steps as we could, both to save us time and to minimize the chance of errors.

Eat our own dog food
Another goal was to actually use the technologies we espouse. We were certainly aware of the irony of a site that promotes open, standards-based computing creating content with a closed-source, proprietary tool such as Microsoft Word. (Note: As of press time, we haven't heard any announcements for a Linux version of Microsoft Office.) One attraction of building tools from XML documents and XSLT style sheets was that it would enable us to show the world that XML and XSLT can do useful work today. Choosing these technologies to manipulate structured data was a no-brainer for us.

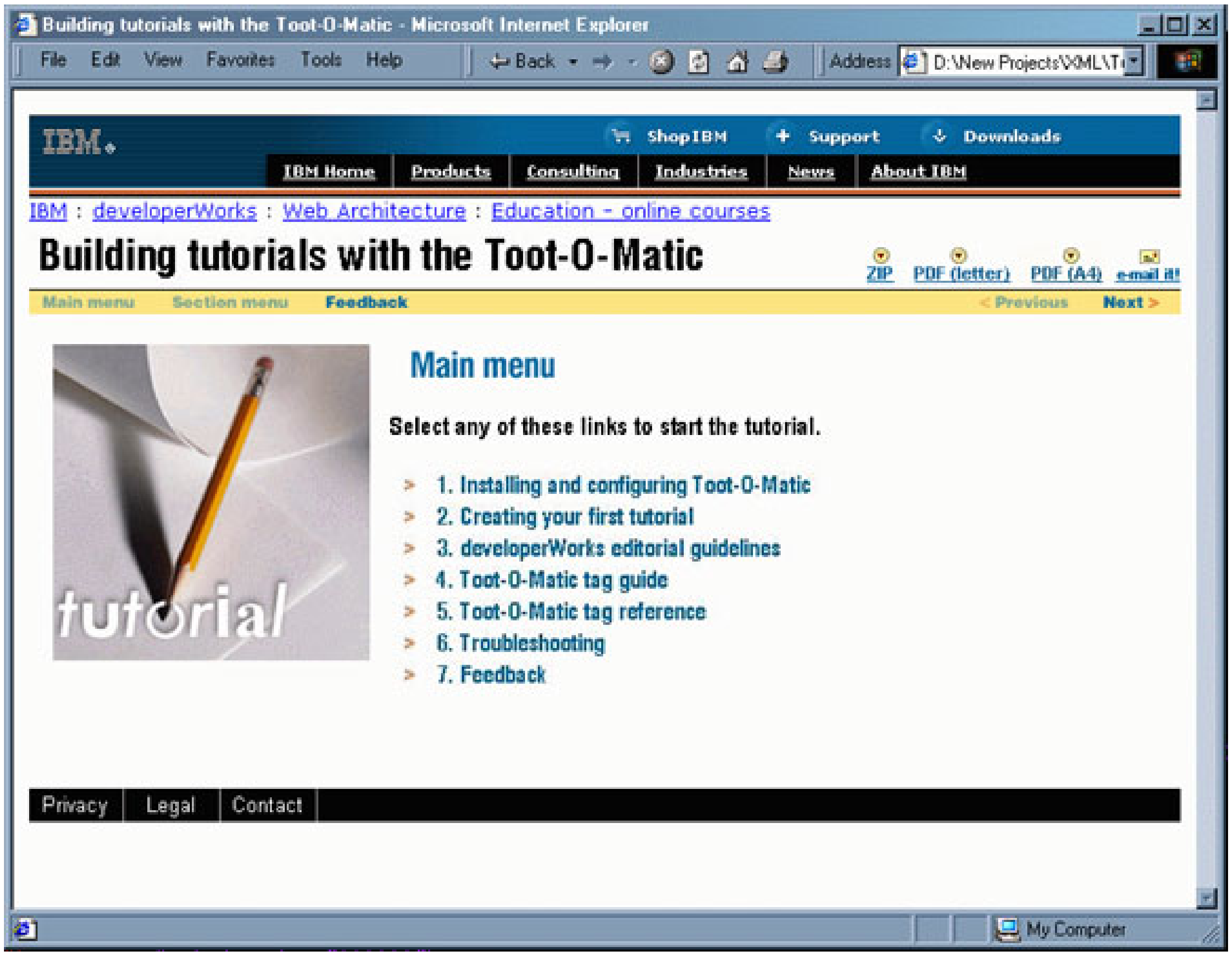
See how much we could accomplish through XSLT style sheets
In achieving the final goal of seeing how much we could do with XSLT, the Toot-O-Matic exercises all of the advanced capabilities of XSLT, including multiple input files, multiple output files, and extension functions. Through the style sheets, it converts a single XML document into:

- A web of interlinked HTML documents
- A menu for the entire tutorial
- A table of contents for each section of the tutorial
- JPEG graphics containing the title text of all sections and the tutorial itself
- A letter-sized PDF file
- An A4-sized PDF file
- A ZIP file containing everything a user needs to run the tutorial on their machine

Tutorial layout
Before going into the XML document structure that evolved over the past 18 months, take a look at the layout of the tutorials. Because we didn't have a legacy of XML-tagged tutorials to begin with, we were able to focus on the end results we wanted and *then* design our XML document structure so that it would be easy for us to transform a conforming XML document into the output formats we needed.

Menu panel
The first thing the user sees is the menu panel. The menu panel looks like this:

Figure 1. Tutorial menu panel


Building tutorials with the Toot-O-Matic - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address D:\New Projects\XML\T
IBM.
Shop IBM
Support
Downloads
IBM Home
Products
Consulting
Industries
News
About IBM
IBM : developerWorks : Web Architecture : Education - online courses
ZIP
PDF (letter)
PDF (A4)
e-mail it!
Main menu
Section menu
Feedback
< Previous
Next >
tutorial
Privacy
Legal
Contact
My Computer

Building tutorials with the Toot-O-Matic

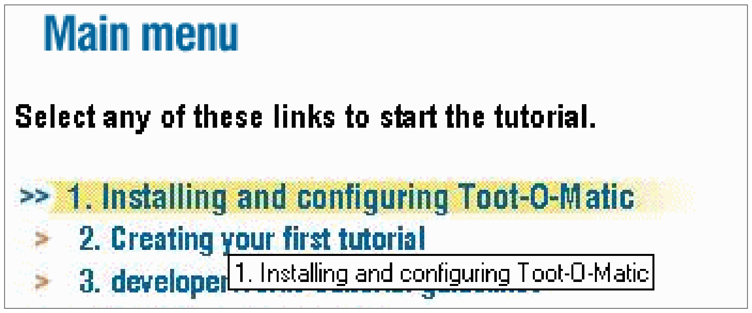
Main menu

Select any of these links to start the tutorial.

- 1. Installing and configuring Toot-O-Matic
- 2. Creating your first tutorial
- 3. developerWorks editorial guidelines
- 4. Toot-O-Matic tag guide
- 5. Toot-O-Matic tag reference
- 6. Troubleshooting
- 7. Feedback

In this example, the string “Building tutorials with the Toot-O-Matic” and all of the section titles (“1. Installing and configuring Toot-O-Matic,” and so on) are JPEG graphics that were created by the tool. If you move the mouse over a given section, its background color changes:

Figure 2. Mouseover effects for a menu item



Main menu

Select any of these links to start the tutorial.

>> 1. Installing and configuring Toot-O-Matic
> 2. Creating your first tutorial
> 3. developer

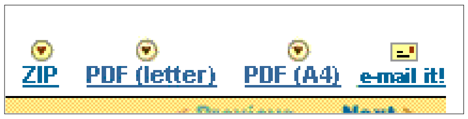
1. Installing and configuring Toot-O-Matic

In Figure 2, notice that the text of the menu item has appeared as a tool tip. This is useful for sight-impaired users, and is consistent with the Web Accessibility Guidelines defined by the W3C and extended by IBM.

Also notice the navigation controls at the top and bottom of the panel. The masthead and footer are defined by IBM's standards for corporate Web sites; this is one of the areas you'll probably want to change when you create your own tutorials with the Toot-O-Matic. The navigation bar contains items such as “Main menu,” “Section menu,” “Feedback,” “Previous,” and “Next.” Although some of those items are disabled on this panel, they appear on all panels in the tutorial.

Above the navigation controls are four icons that allow users to download alternate versions of the tutorial or to e-mail the tutorial to a friend:

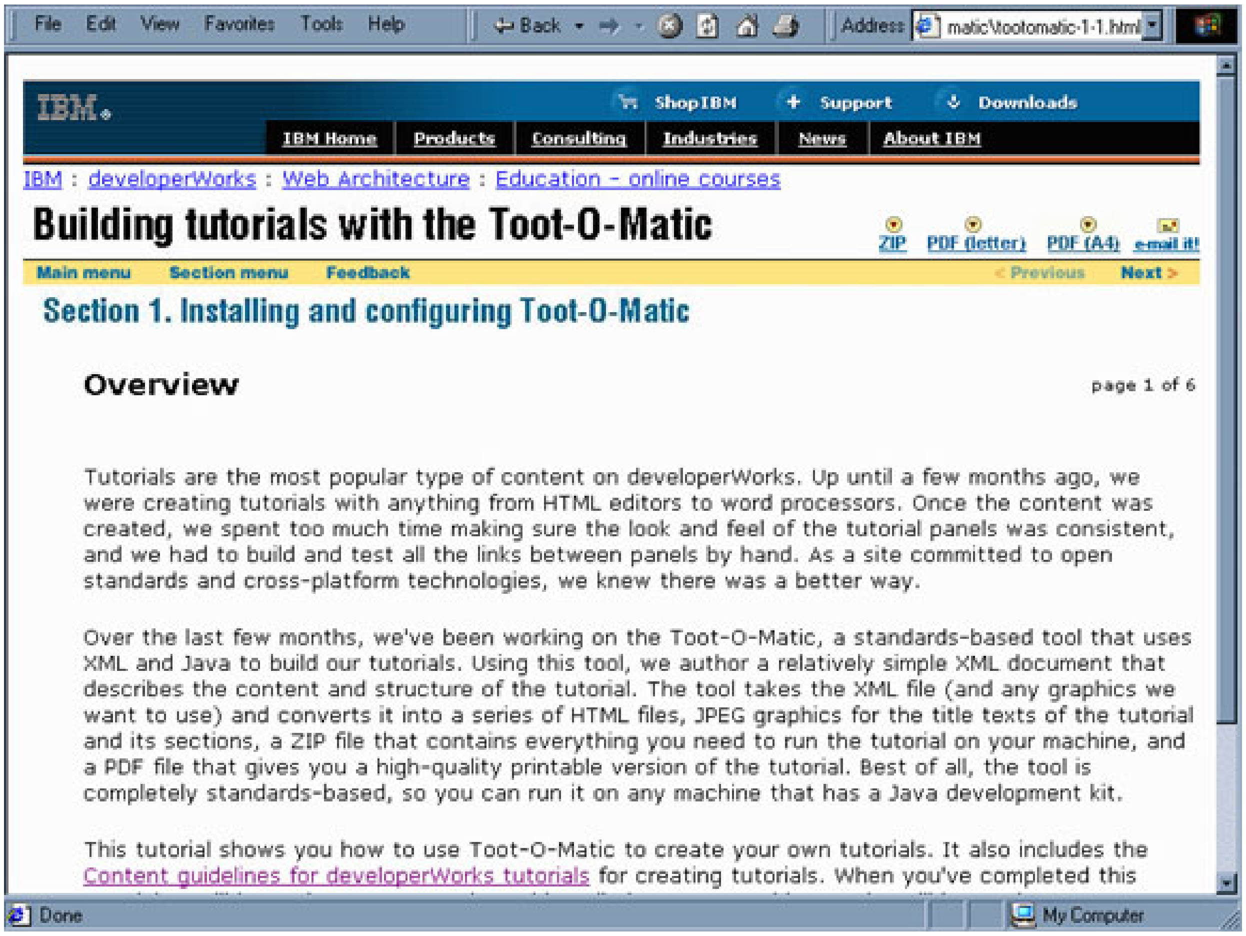
Figure 3. Tutorial icons



From left to right, the icons allow users to download a ZIP file that contains all of the files necessary to run the tutorial locally, a letter-sized PDF file, an A4-sized PDF file, and e-mail a note to a friend that recommends this tutorial. All of these icons appear on every panel in the tutorial, and their placement and their associated links are generated by the Toot-O-Matic.

Individual panels
An individual panel looks like this:

Figure 4. Tutorial information panel



Building tutorials with the Toot-O-Matic

Section 1. Installing and configuring Toot-O-Matic

Overview

page 1 of 6

Tutorials are the most popular type of content on developerWorks. Up until a few months ago, we were creating tutorials with anything from HTML editors to word processors. Once the content was created, we spent too much time making sure the look and feel of the tutorial panels was consistent, and we had to build and test all the links between panels by hand. As a site committed to open standards and cross-platform technologies, we knew there was a better way.

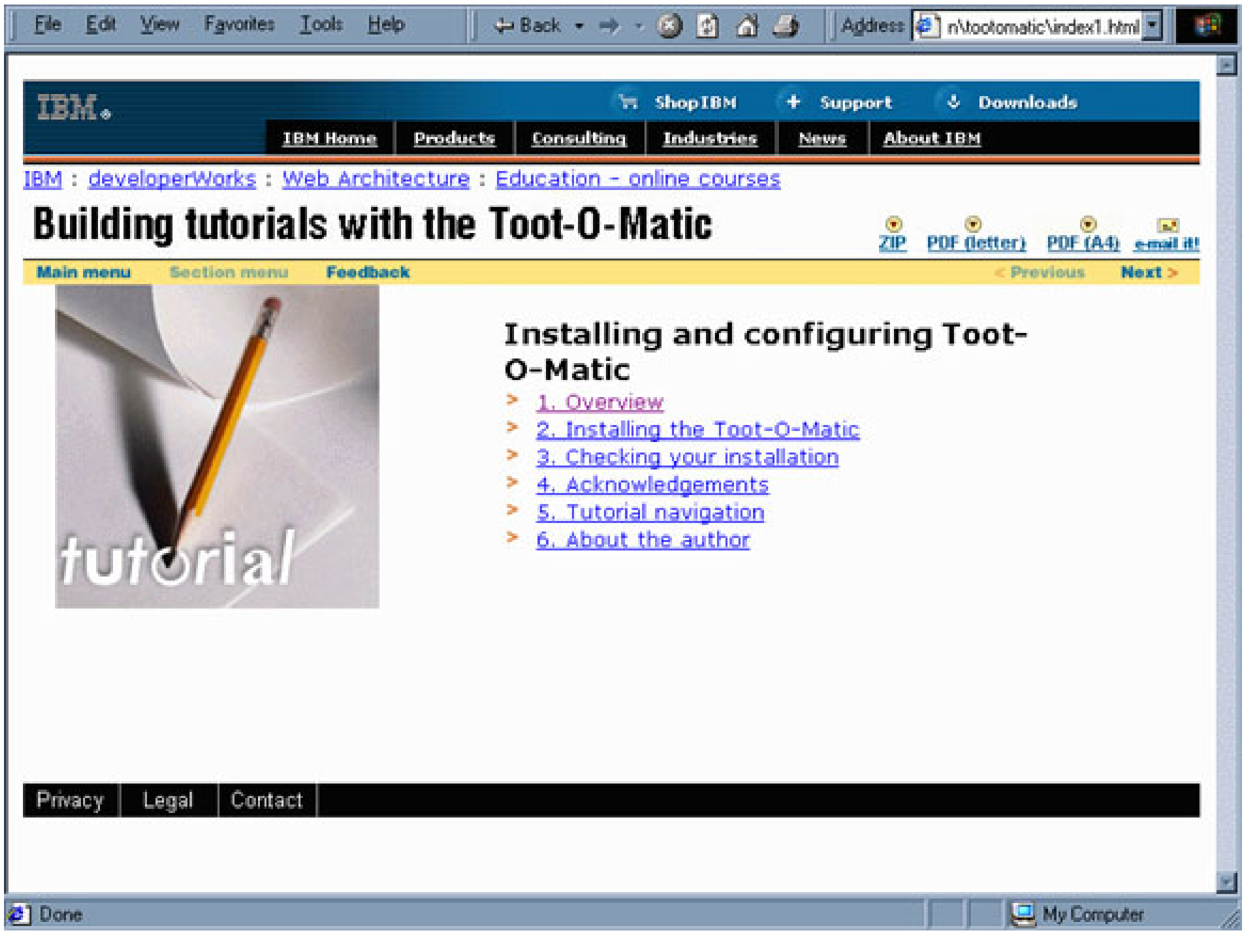
Over the last few months, we've been working on the Toot-O-Matic, a standards-based tool that uses XML and Java to build our tutorials. Using this tool, we author a relatively simple XML document that describes the content and structure of the tutorial. The tool takes the XML file (and any graphics we want to use) and converts it into a series of HTML files, JPEG graphics for the title texts of the tutorial and its sections, a ZIP file that contains everything you need to run the tutorial on your machine, and a PDF file that gives you a high-quality printable version of the tutorial. Best of all, the tool is completely standards-based, so you can run it on any machine that has a Java development kit.

This tutorial shows you how to use Toot-O-Matic to create your own tutorials. It also includes the Content guidelines for developerWorks tutorials for creating tutorials. When you've completed this

Most of the panels in a tutorial use this design. Notice that the panel contains the text “page 1 of 6” at the top. The navigation bar also has active links to the main menu and the section menu.

Section menu
Clicking the “Section menu” link displays a list of all the titles in the current section:

Figure 5. Tutorial section index panel



Building tutorials with the Toot-O-Matic

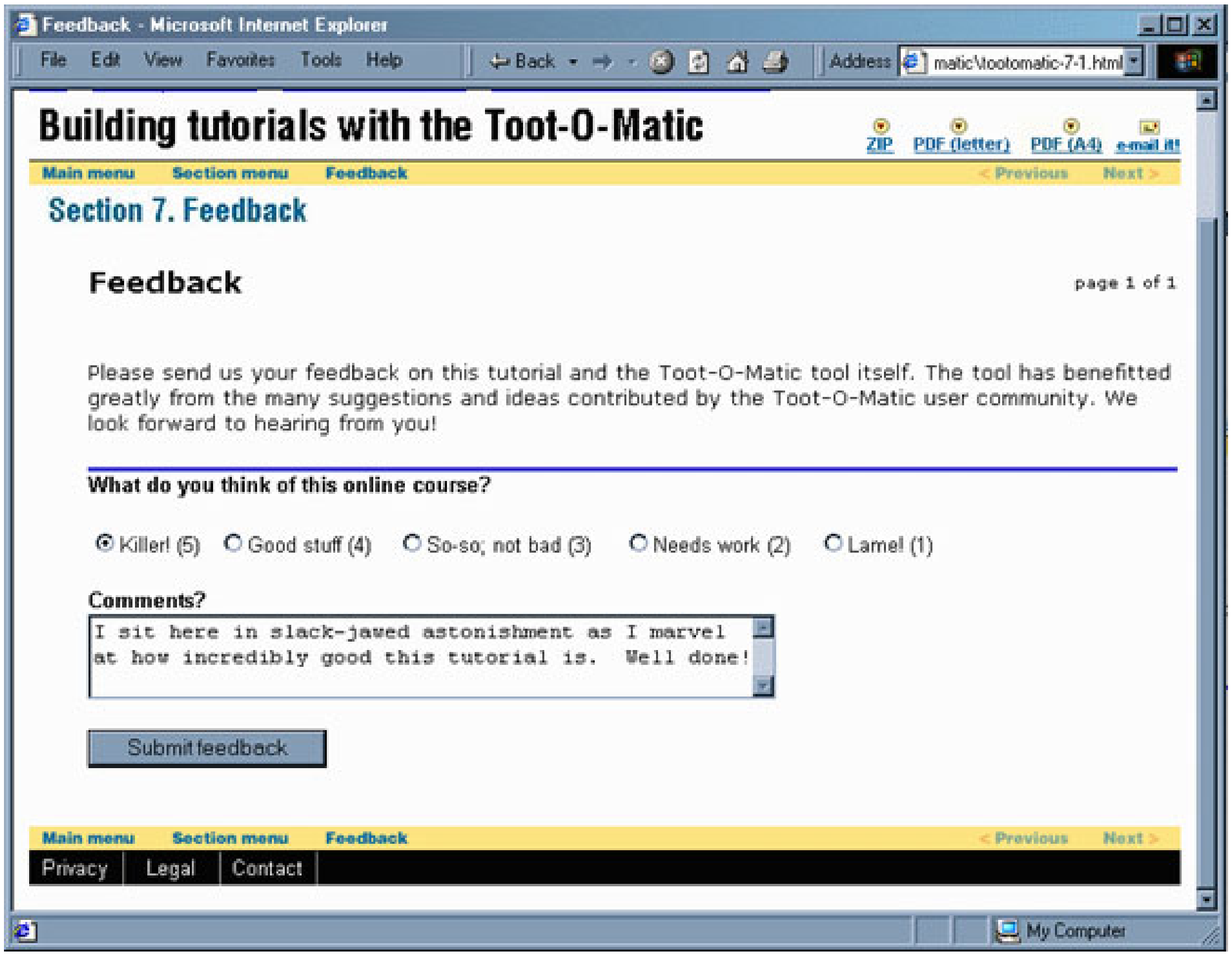
Installing and configuring Toot-O-Matic

- 1. Overview
- 2. Installing the Toot-O-Matic
- 3. Checking your installation
- 4. Acknowledgements
- 5. Tutorial navigation
- 6. About the author

You can click on any of the panel titles to go directly to that panel. Through the magic of style sheets, the title of each panel automatically appears, along with the correct hyperlink to each panel.

Feedback panel
The Toot-O-Matic also generates a feedback panel automatically. The panel contains a brief paragraph, followed by a feedback form. A user's comments and opinions are automatically entered into our feedback database when the user clicks the “Submit feedback” button. Here's what a feedback panel looks like:

Figure 6. Feedback panel


Feedback - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address matic\tootomatic-7-1.html
ZIP PDF (letter) PDF (A4) e-mail it!
Main menu Section menu Feedback
< Previous Next >
Privacy Legal Contact
My Computer

Building tutorials with the Toot-O-Matic

Section 7. Feedback

Feedback

page 1 of 1

Please send us your feedback on this tutorial and the Toot-O-Matic tool itself. The tool has benefitted greatly from the many suggestions and ideas contributed by the Toot-O-Matic user community. We look forward to hearing from you!

What do you think of this online course?

[✓ Killer! (5)] Good stuff (4) So-so; not bad (3) Needs work (2) Lame! (1)

Comments?

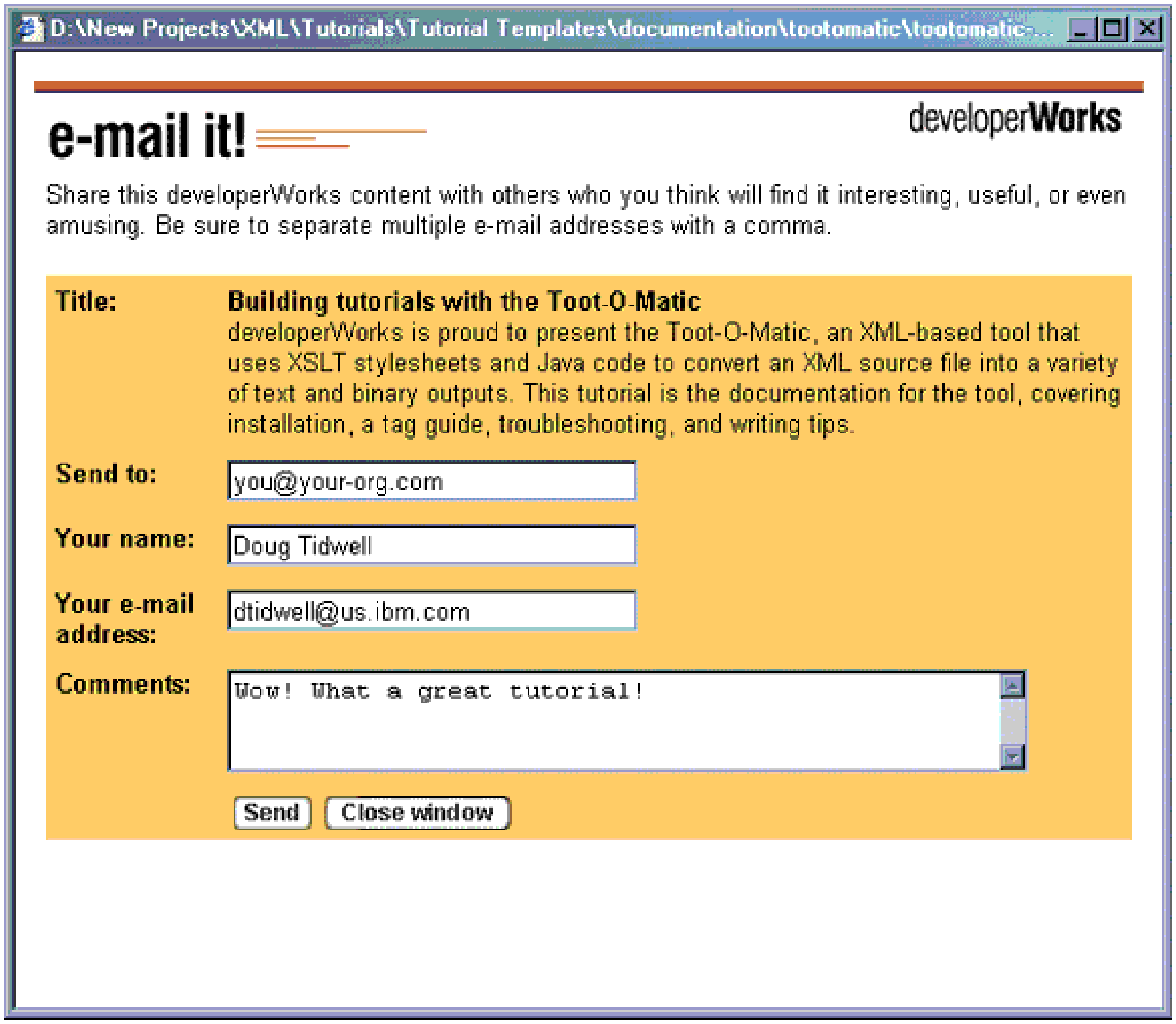
```
I sit here in slack-jawed astonishment as I marvel
at how incredibly good this tutorial is.  Well done!
```

Submit feedback

Once a given panel is identified as the feedback panel, a link to that panel is displayed on the navigation bar of every panel in the tutorial.

E-mail panel
A relatively recent addition is the “E-mail a friend” panel. This allows users to share their enthusiasm for a given tutorial with their friends. Clicking the “e-mail it!” icon displays a panel like this:

Figure 7. E-mail panel


D:\New Projects\XML\Tutorials\Tutorial Templates\documentation\tootomatic\tootomatic-...
e-mail it!
developerWorks

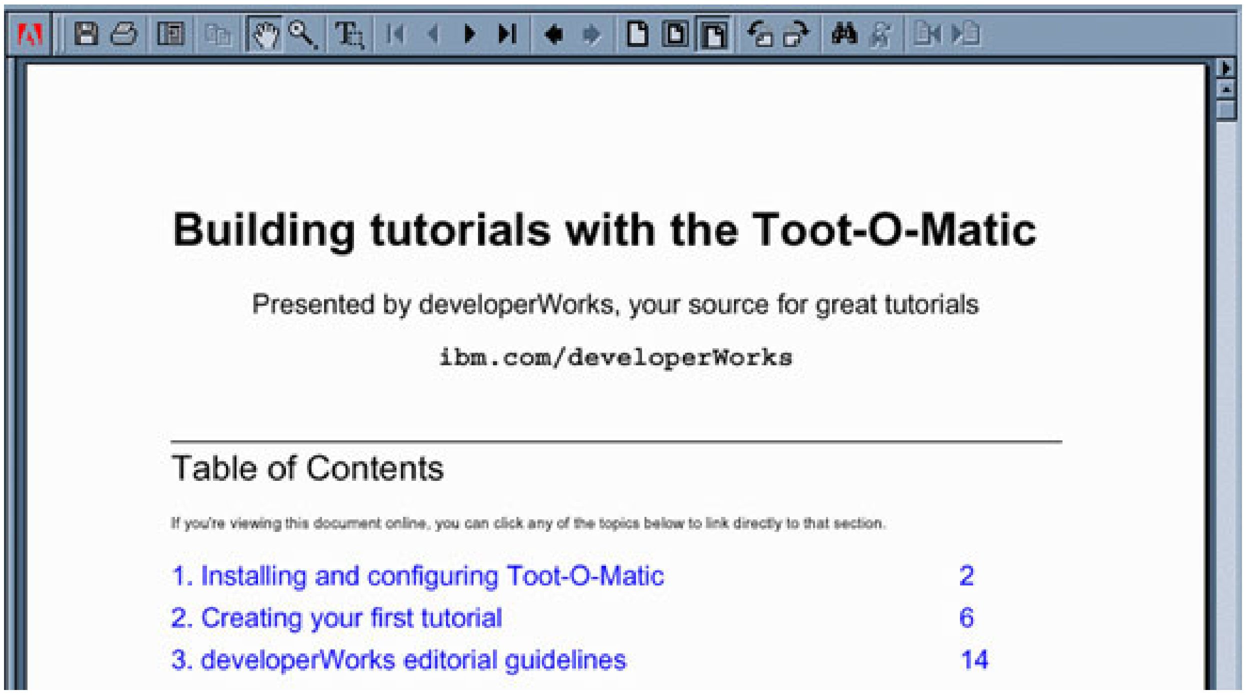
Share this developerWorks content with others who you think will find it interesting, useful, or even amusing. Be sure to separate multiple e-mail addresses with a comma.

Title:	**Building tutorials with the Toot-O-Matic** developerWorks is proud to present the Toot-O-Matic, an XML-based tool that uses XSLT stylesheets and Java code to convert an XML source file into a variety of text and binary outputs. This tutorial is the documentation for the tool, covering installation, a tag guide, troubleshooting, and writing tips.
Send to:	you@your-org.com
Your name:	Doug Tidwell
Your e-mail address:	dtidwell@us.ibm.com
Comments:	Wow! What a great tutorial!

ZIP files
The ZIP file version of the tutorial contains all of the HTML files necessary to view the tutorial, as well as all of the supporting graphics and other files. When the Toot-O-Matic builds this file, it includes all of the generated HTML and JPEG files, as well as any resources referenced in the tutorial XML source file itself.

PDF files
The PDF version of the tutorial supports high-quality printed output for users that want to read the tutorial offline. The first page of the tutorial features the title of the tutorial and a table of contents:

Figure 8. The first page of the PDF file



Building tutorials with the Toot-O-Matic

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

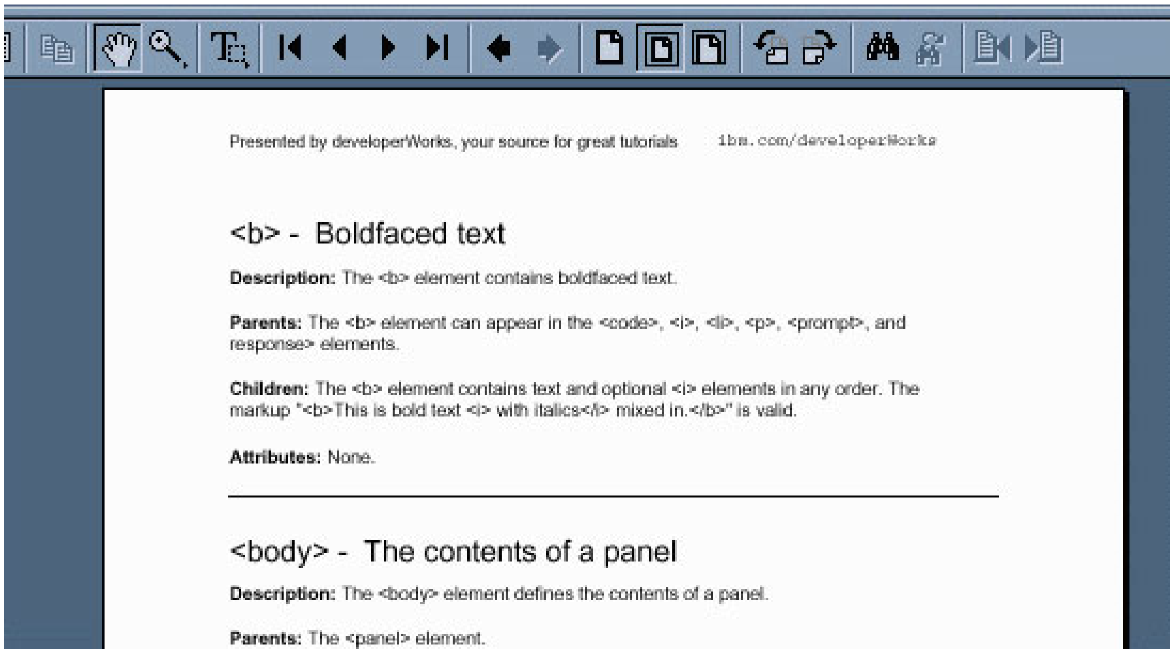
If you're viewing this document online, you can click any of the topics below to link directly to that section.

1. Installing and configuring Toot-O-Matic	2
2. Creating your first tutorial	6
3. developerWorks editorial guidelines	14

In the table of contents, both the section titles and the page numbers are hyperlinks; if you're viewing the PDF file online, you can use those links to go directly to a particular section. If you're reading a printout of the PDF file, the table of contents is still useful. Besides the links in the table of contents, any cross-references in the tutorial itself and any references to Web pages are hyperlinks as well.

Pages in the body of the tutorial feature the text and illustrations of each panel, separated on the page with a horizontal line:

Figure 9. A page from the body of the tutorial



Presented by developerWorks, your source for great tutorials ibm.com/developerWorks

<b> - Boldfaced text

Description: The <b> element contains boldfaced text.

Parents: The <b> element can appear in the <code>, <i>, <li>, <p>, <prompt>, and response> elements.

Children: The <b> element contains text and optional <i> elements in any order. The markup "<b>This is bold text <i> with italics</i> mixed in.</b>" is valid.

Attributes: None.

<body> - The contents of a panel

Description: The <body> element defines the contents of a panel.

Parents: The <panel> element.

All of the formatting and layout issues in the PDF files are handled by the Toot-O-Matic.

XML document design
Now that I've covered how the developerWorks tutorials appear in all their various forms, I'll discuss the structure of the XML documents that become our tutorials. To start with, there were some obvious structural principles we used:

- A `<tutorial>` should contain a single `<title>` and one or more `<section>`s.
- A `<section>` should contain a single `<title>` and one or more `<panel>`s.
- A `<panel>` should contain a single `<title>` and a `<body>`, which would in turn contain markup for the

contents of the one or two columns of the panel.

These decisions were obvious because of the way our tutorials are structured.

Individual panels
Our first major design decision was determining how an individual <panel> should be marked up. We settled on the following structure:

Listing 1. Tutorial document structure

```
<panel>
  <title>Title of the panel</title>
  <body>
    <image-column> -or- <example-column> (one or the other or neither)
    <text-column>
      Basic HTML markup (<p>, <ol>, <li>, <b>, <i>, <u>, <a>, etc.)
    </text-column>
  </body>
</panel>
```

This structure allows tutorial writers to use mostly tags they're familiar with, while allowing us to easily transform their content into a variety of formats. The `<image-column>` and `<example-column>` elements define the contents of the left-hand column, if there is one. The Toot-O-Matic documentation (a tutorial that comes with the Toot-O-Matic package) describes all of the elements and attributes supported by the tool; most of them are similar to the HTML elements you know and love.

Filenames
The goal was to convert a single XML file into a web of interlinked HTML documents. To do that, we needed some standard way of naming all of the files created as output. Adding the filename attribute to the `<tutorial>` element defines the root of the filename. If the filename attribute is "xyz," then the HTML files in section 1 are named `xyz-1-1.html, xyz-1-2.html`, and so on, and the files in section 2 are named `xyz-2-1.html, xyz-2-2.html`, and the like. For linking, if I'm in the fourth `<panel>` in the second `<section>`, I know the **Previous** link should point to the file `xyz-2-3.html`. If the current panel isn't the last, then the **Next** link should point to the file `xyz-2-5.html`. The section index for the second section is named `index2.html`, so the Section menu link should point to that file. When we create the main menu panel, if there are seven `<section>`s in our tutorial, we need to create links to `xyz-1-1.html, xyz-2-1.html`, right on through `xyz-7-1.html`.

For the names of the files referenced by the ZIP and PDF icons, we use the `filename` attribute again. Continuing with the previous example, the ZIP file would be named `xyz.zip`, and the two PDF files would be named `xyz-ltr.pdf` and `xyz-a4.pdf`. Knowing the value of the `filename` attribute allows us to build these links on the masthead of every HTML file in the tutorial. Having a consistent naming convention makes it possible to build the links between panels.

Feedback panel
The feedback panel is determined by the first <panel> that contains the <feedback-form> element. That element looks like this:

Listing 2. The XML element that generates the feedback panel

```
<feedback-form
  action-url="http://www9.software.ibm.com/dworks/ratings.nsf/
    RateOnlineCourse?CreateDocument"
  zone="Web"
  redirect-url="http://www.ibm.com/developer/thankyou/feedback-java.html" />
```

If you want to use the Toot-O-Matic for your own purposes, change the `action-url` and `redirect-url` attributes to match your site.

E-mail panel

The recently introduced e-mail panel required a couple of new fields, both of which were added as attributes to the `<tutorial>` element. Here's what the XML source looks like:

Listing 3. The attributes used to create the e-mail panel

```
<tutorial filename="tootomatic"
. . .
  email-link="http://xyz.ibm.com/dW-tutorials/education/tootomatic"
  abstract="developerWorks is proud to present the Toot-O-Matic,
  an XML-based tool that uses XSLT style sheets and Java code to convert
  an XML source file into a variety of text and binary outputs.
  This tutorial is the documentation for the tool, covering installation,
  a tag guide, troubleshooting, and writing tips.">
```

In this listing, the `email-link` and `abstract` attributes are used by the e-mail panel. To build the e-mail panel, we use a standard JavaScript file, `emailfriend.js`, that manages the panel. Here's the HTML we generate to make the e-mail panel work:

Listing 4. Generated HTML markup to invoke the e-mail panel

```
<a href="javascript:void newWindow()" border="0">
  <img alt="E-mail this tutorial to a friend" border="0"
   src="../i/icon-email.gif">
</a>
```

(I tidied up this listing so you can better see what's going on. The actual HTML files generated by the Toot-O-Matic tend to have much less whitespace.)

A quick word about the directory structure: The Toot-O-Matic generates all of its files in a given subdirectory, specified either on the command line or in the `<tutorial>` element. All common resources (`emailfriend.js`, standard mouseover graphics, and so on) are stored in the `../i` directory. If you store multiple Toot-O-Matic-generated tutorials on a single machine, the common files will be stored only once.

XSLT source code
Now that I've discussed the design issues we resolved as we defined the XML document structure, I'll talk about how the XSLT style sheets transform the XML documents into the results we want. First of all, we use the XSLT mode attribute to process the same structured information in multiple ways:

Listing 5. Using the XSLT mode attribute

```
<xsl:template match="/">
    <xsl:apply-templates select="tutorial" mode="build-main-index"/>
    <xsl:apply-templates select="tutorial" mode="build-section-indexes"/>
    <xsl:apply-templates select="tutorial" mode="build-individual-panels"/>
    <xsl:apply-templates select="tutorial" mode="generate-graphics"/>
    <xsl:apply-templates select="tutorial" mode="generate-pdf-file">
      <xsl:with-param name="page-size" select="'letter'"/>
    </xsl:apply-templates>
    <xsl:apply-templates select="tutorial" mode="generate-pdf-file">
      <xsl:with-param name="page-size" select="'a4'"/>
    </xsl:apply-templates>
    <xsl:apply-templates select="tutorial" mode="build-zip-file"/>
  </xsl:template>
```

This example shows the processing of the same basic data in several different ways. We use the mode `build-main-index` to build the main index page for the tutorial, we use the `build-individual-panels` mode to build the individual panels, and so on. Notice that the `generate-pdf-file mode` uses the `page-size` parameter to set the page size correctly (we generate both a letter-sized and A4-sized PDF files).

Generating the main menu panel
The main menu panel consists of a standard header and footer, with a list of all the sections of the tutorial in between. Clicking any of the section titles takes you to the first panel in that section. To enhance the visual appeal of the panel, generated graphics and mouseover effects are used to display the panel title.

The style sheet that generates the list of sections is straightforward. The header and footer are generated from boilerplate text; the list of sections is generated with an `<xsl:for-each>` element:

Listing 6. XPath statement to generate menu panel links

```
<xsl:for-each select="section">
  <a>
    <xsl:attribute name="href">
      <xsl:value-of select="concat($fn, '-', position(), '-1.html')"/>
    </xsl:attribute>
    ...
    <img width="335" height="26" border="0">
      ...
      <xsl:attribute name="src">
        <xsl:value-of
          select="concat('imagemaster/menu-', position(), '.jpg')"/>
      </xsl:attribute>
    </img>
  </a>
  <br/>
</xsl:for-each>
```

(To keep the example brief, I removed some of the XPath expressions that generate the various attributes of the anchor and image tags.) As an example, for the first section **`(position()=1)`**, the style sheet generates the following HTML:

Listing 7. Sample menu panel link

```
<a href="xyz-1-1.html" onMouseOut="iOut('menu1');"
   onMouseOver="iOver('menu1'); self.status=menu1blurb; return true;">
  <img border="0" height="26" name="menu1"
       src="imagemaster/menu1.jpg" width="335"/>
</a>
<br/>
```

In the listing above, all of the 1s in boldface were generated by the XPath `position()` function.

Generating the individual HTML panels
The next task is to generate the individual HTML panels that comprise the tutorial. The header and footer of the tutorial are generated by boilerplate, and most of the tags in the body of the panel are virtually identical to their HTML counterparts. The most interesting thing about generating the individual panels is that we use an XSLT extension to redirect the output of the transformation into different files.

We do that with the text redirection extension shipped with the Apache XML Project's Xalan style sheet engine. Before we use the extension, we have to declare it in the `<xsl:stylesheet>` element, the root element of the style sheet:

Listing 8. Style sheet extension definition

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  xmlns:redirect="org.apache.xalan.xslt.extensions.Redirect"
  extension-element-prefixes="redirect">
```

In this example, `org.apache.xalan.xslt.extensions.Redirect` is the name of the Java class that implements the extension, and `redirect` is the namespace prefix used to invoke the extension. Once we've declared the extension, we can use it to pipe the transformation output to different files. A wonderful feature of the `Redirect` extension is that we can use XPath expressions to generate the file names:

Listing 9. Style sheet extension invocation

```
<redirect:write select="concat($fn, '-',
                               $sectionNumber, '-',
                               position(), '.html')">
  <!-- other processing goes here -->
</redirect:write>
```

Assuming the prefix is `xyz`, the XPath `concat` function call above creates the file name `xyz-5-4.html` for the fourth `<panel>` in the fifth `<section>` of a `<tutorial>`. (To streamline processing, the filename attribute of the `<tutorial>` tag is stored in the variable `fn`, and the `position()` of the current `<section>` is stored in the variable `sectionNumber`.) We generate link points and references throughout the processing of the XML source document. That ensures that references remain correct and consistent as the source document changes.

Generating the section indexes
To generate the section index, we create an HTML file with an ordered list of all of the `<panel>`s in a given `<section>`. Retrieving all of the panel titles uses a simple `<xsl:for-each>` statement; because we generate all the file names dynamically, we can create hyperlinks to all of the panels as well. Here's a brief excerpt of the XSLT template we use for that task:

Listing 10. Generating a section index

```
<ol>
  <xsl:for-each select="panel">
    <li>
      <a>
        <xsl:attribute name="href">
          <xsl:value-of
        select="concat($fn, '-', $sectionNumber, '-', position(), '.html')"/>
        </xsl:attribute>
        <xsl:value-of select="title"/>
      </a>
    </li>
  </xsl:for-each>
</ol>
```

In the example above, the `concat` function call we used to generate the file name is used to generate the value of the `href` attribute on the `<a>` tag. Inside the `<a>` tag itself, we use an `<xsl:value-of>` element to retrieve the `<title>` of the current panel.

Generating the PDF files
Converting the XML document to an XSL Formatting Objects (XSL-FO) stream is fairly straightforward as well. Our printed layout consists of the graphics and text from the tutorial, combined with page numbers, headers, and footers to create high-quality printed output.

Although as of this writing (May 2001) the XSL-FO specification isn't finished, we can use the formatting objects currently supported by the Apache XML Project's FOP (Formatting Objects to PDF) tool. Here's how a couple of paragraphs might look after they have been converted to formatting objects:

Listing 11. Sample formatting objects

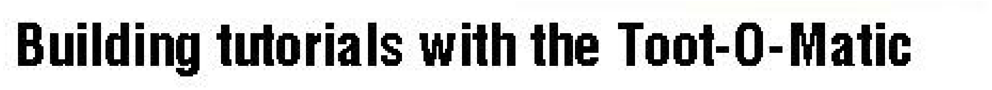
```
<fo:block font-size="8pt" line-height="10pt"
  text-align-last="end" space-after.optimum="8pt">
  page 1 of 14
</fo:block>
<fo:block font-size="16pt" line-height="19pt" font-weight="bold"
space-after.optimum="12pt">
  Introduction to JavaServer Pages
</fo:block>
<fo:block space-after.optimum="6pt">
  In today's environment, most Web sites want to display dynamic
  content based on the user and the session. Most content, such
  as images, text, and banner ads, is most easily built with
  HTML editors. So we need to mix the "static" content of HTML
  files with "directives" for accessing or generating dynamic
  content.
</fo:block>
<fo:block space-after.optimum="6pt">
  JavaServer Pages meet this need. They provide server-side
  scripting support for generating Web pages with combined
  static and dynamic content.
</fo:block>
```

Generating the JPEG files
For any tutorial, we generate graphics that highlight certain portions of the tutorial. First of all, we create an image for the masthead graphic on the title page. The text of the masthead comes from the `/tutorial/title` element. As an example, the markup

```
<tutorial>
  <title>Building tutorials with the Toot-O-Matic</title>
```

generates a `masthead.jpg` file that looks like this:

Figure 10. Generated masthead graphic



To create this text, we use a Java extension that converts the text of the `<title>` element into a JPEG file:

Listing 12. Invoking the JPEG-creating extension

```
<xsl:for-each select="/book/chapter">
  <xsl:choose>
    <xsl:when test="function-available('jpeg:createJPEG')">
      <xsl:value-of
       select="jpeg:createJPEG(title, 'bg.jpg',
       concat('title', position(), '.jpg'),
       'Swiss 721 Bold Condensed', 'BOLD', 22, 52, 35)"/>
      <img>
        <xsl:attribute name="src">
          <xsl:value-of select="concat('title', position(), '.jpg')"/>
        </xsl:attribute>
      </img>
      <br />
    </xsl:when>
    <xsl:otherwise>
      <h1><xsl:value-of select="title"/></h1>
    </xsl:otherwise>
  </xsl:choose>
</xsl:for-each>
```

Generating the ZIP file
The final task is to generate the ZIP file itself, which is handled with an extension as well. This is another area in which the file naming convention simplifies things. We pass the extension function the root of the XML source file:

Listing 13. Generating the ZIP file

```
<xsl:template match="tutorial" mode="build-zip-file">
  <xsl:choose>
    <xsl:when test="function-available('zip:buildZip')">
      <xsl:value-of select="zip:buildZip(.)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:message terminate="yes">
        Sorry, we can't build the zip file.
      </xsl:message>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Notice that we had to pass only the root element when we invoked the `buildZip` function. The `buildZip` function takes the node we gave it (the document node), and attempts to create a ZIP file and populate it with all of the required files. Some of those files are part of the standard list of resources used by every Toot-O-Matic tutorial, while others are resources (JPEGs, GIFs, and the like) referenced by the tutorial. The HTML files added are determined by the structure of the XML source file. If the tutorial has four or more sections, and the fourth section contains eight panels, and the filename prefix is `tom`, the `buildZip` function needs to add files `tom-4-1.html, tom-4-2.html`, and so on, to the ZIP archive.

Summary
This discussion of the Toot-O-Matic tool illustrates the full range of outputs that you can generate from a single XML file. The structure of our original XML documents enables us to convert flat textual information into a number of different formats, all of which work together to deliver a single piece of content in a variety of interesting and useful ways. Using this tool, we have shortened and streamlined our development process, making it easier, faster, and cheaper to produce our tutorials. Best of all, everything we've described here is based on open standards and works on any Java-enabled platform. The Toot-O-Matic tool shows how a simple, inexpensive development project can deliver significant results.

Resources

- Download the free Toot-O-Matic package, which includes:
  - The sets of XML, XSL, Java, and .jar files that you'll download to your hard drive to run the tool. You'll need to be running the Java 2 SDK, 1.3.0_02.
  - Sample tutorial files and Toot-O-Matic documentation.
  - A readme file.
- Take the Building tutorials with the Toot-O-matic tutorial to learn how to use Toot-O-Matic to create your own tutorials.
- Join the Toot-O-Matic forum on developerWorks to ask a question or make a comment about the tool.
- Check out some other articles on developerWorks by Doug Tidwell that might be useful background as you work with the Toot-O-Matic:
  Java developers: Fill your XML toolbox: The Toot-O-Matic package comes with all of the files you need to build tutorials. If you want to start working with the various technologies used by the Toot-O-Matic, this article will help you install the tools you'll need.
- Converting XML into HTML, Converting XML into SVG, and Converting XML into PDF: The three-part series takes several XML documents and converts them into a variety of other formats. If you're just getting started with XSLT, these articles will help.
- What kind of language is XSLT? by Michael Kay, creator of the open-source Saxon XSLT processor, gives a good overview of XSLT as a language, covering its design goals and including some sample style sheet examples.
- For details about Xalan and Xerces, the open-source XSLT processor and XML parser used in the Toot-O-Matic, go directly to the source at the Apache XML Project home page.
- Take a 17-question survey about your XML development activities and contribute to the direction of XML research in IBM's labs.
- If you want to know how IBM's WebSphere Application Server (WAS) supports XML development, see this technical background info on XML in the WAS Advanced Edition 3.5 online help.

Acknowledgements
At various times, the developerWorks tutorial team has included Tom Coppedge, Jeanette Fuccella, Leah Ketring, Jeanne Murray, Christine Stackel, Doug Tidwell, Jackie Wheeler, and Janet Willis. The Toot-O-Matic was shaped by their ideas, as well as by excellent suggestions from Lou Shannon, Gretchen Moore, and other Toot-O-Matic users.

About the author

MC Dug-T is developerWorks' Minister of Science, droppin' the XML, Java, and Web services 411 on the public. In his travels, he gets mad props from his peeps worldwide for the stone-cold, stoopid-fresh style sheets he leaves behind. All his mad-phat nollidge will soon be published by O'Reilly and Associates in the Strictly Non-Fiction book XSLT (ISBN 0596000537, pre-order your copy today at amazon.com) which will then start slayin' soft-sellin' suckas at tha local booksella. Discussing the book in a recent dW interview, he boasted, “I'm gonna empty mah dome into one supa-fly tome.”

For relaxation, he likes to put his hands up in the air, and in his words, “wave 'em around like I just don't care.” When not chillin' with his worldwide XML krew, he maxes at the crib in Raleigh with his wife, cooking teacher CT-ONE, and their six-year-old shortie, Lily the Flayva Princess. You can send him a shout-out at dtidwell@us.ibm.com.

Discuss e-mail it!

What do you think of this article?

○ Killer! (5) ○ Good stuff (4) ○ So-so; not bad (3) ○ Needs work (2) ○ Lame! (1)

Comments? Care to share your comments? Try the Forum.

Submit feedback