Extracted from:

XSLT Jumpstarter

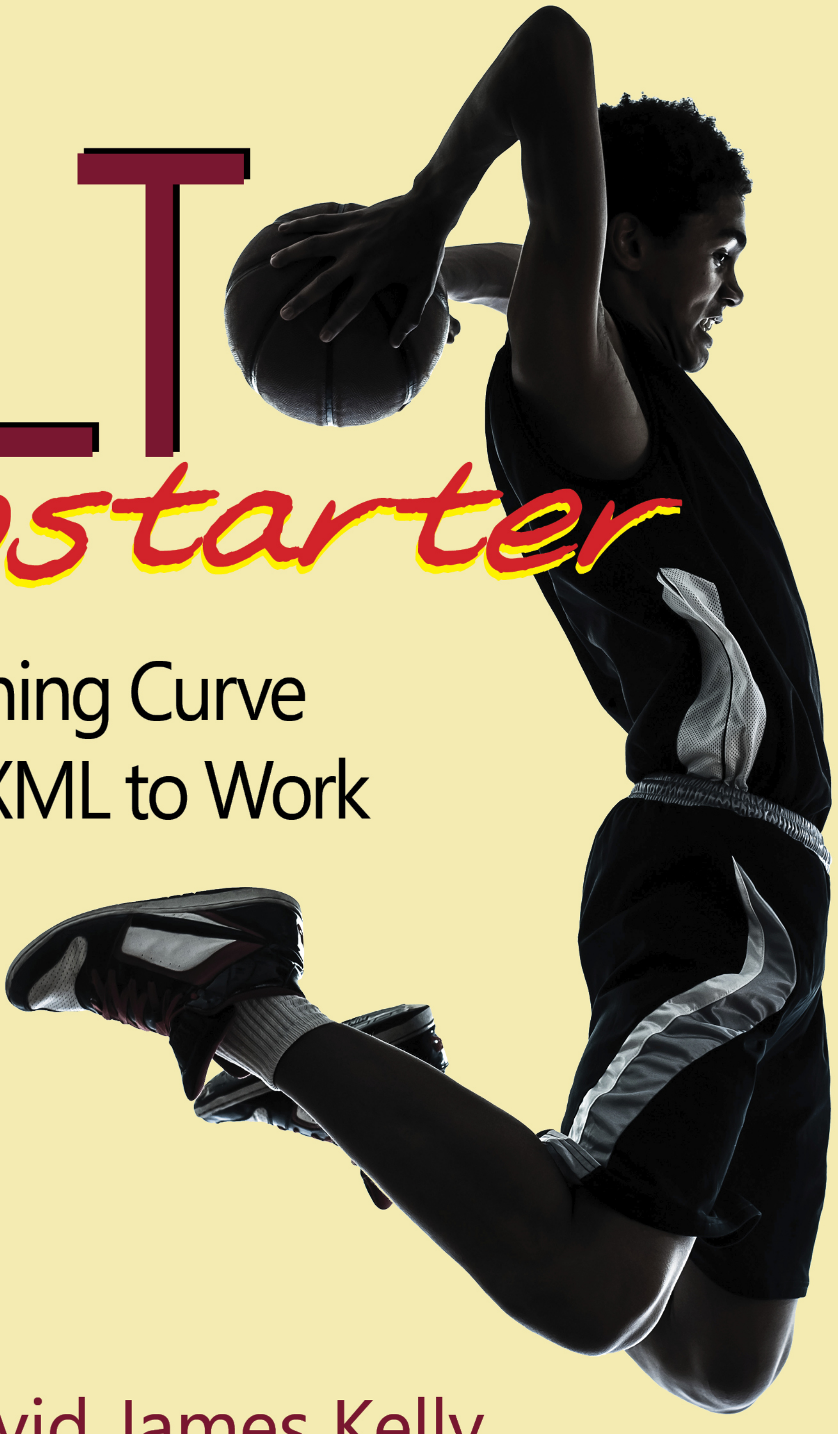
Level the Learning Curve and Put Your XML to Work

Peloria Press

Raleigh, North Carolina

XSLT Jumpstarter

Level the Learning Curve and Put Your XML to Work



David James Kelly

Foreword by Dave Thomas

Reaping the Flow: Built-in Template Rules

As we've seen, one way of getting text out of an XML file is to use the <xsl:apply-templates> instruction and rely on the built-in template rules to retrieve the text that is in the XML source document.

Let's set up a simple XML document and XSLT template to retrieve its text. Here's the XML:

handlingcontent/caution1.xml
```
<?xml version='1.0' encoding='UTF-8'?>
<?xml-stylesheet type="text/xsl" href="caution1.xsl"?>
<caution type="Intergalactic">
  <paragraph>The potential for intergalactic broadcast while using
    the <product>AZGuard Protaxis</product> unit
    has not yet been determined.</paragraph>

  <paragraph>Please avoid transmitting information that could place
    the human race at risk for alien invasion.</paragraph>
</caution>
```

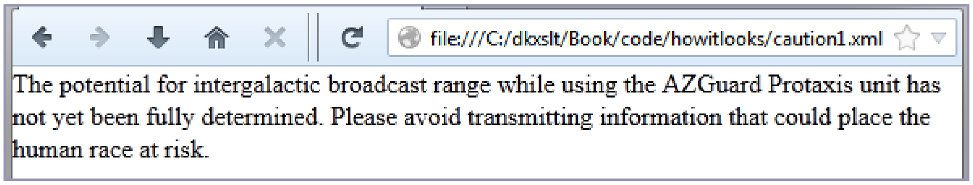
Figure 11—Example XML for a cautionary note

Here's the stylesheet that will give us all the text in the <caution> tag:

handlingcontent/caution1.xsl
```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/caution">
  <xsl:apply-templates/>
</xsl:template>
</xsl:stylesheet>
```

Since the <?xml-stylesheet/> processing instruction in the XML file points to the XSLT stylesheet, we could just open the XML file in a browser to see the results. Remember that the XSLT stylesheet should be in the same directory as the XML file for our examples.

Here's the result we should see: an unformatted string that concatenates all the text in the file:


file:///C:/dkxslt/Book/code/howitlooks/caution1.xml

The potential for intergalactic broadcast range while using the AZGuard Protaxis unit has not yet been fully determined. Please avoid transmitting information that could place the human race at risk.

We’ve seen this before, and it’s not particularly useful. This is a good way to get the text out in the order it appears in the document, if that’s all we’re interested in. All we need to do is create a template to match something and put an <xsl:apply-templates> in it. It will retrieve all the text in the element we matched, including the text of any descendant tags.

If there are no descendant tags, this is a safe way to output the text, but usually we’re going to need more than that. We’re going to want to format the text, add stuff to it, maybe even leave things out. Let’s see if we can get some help with that.

Fine Tuning the Output with <xsl:value-of>

We’ll heed the famous admonition of *The Elements of Style* by Strunk and White: "Omit needless words." Instead of using <xsl:apply-templates> and the indiscriminate reaping machine of the default template, let’s use something a little pickier: <xsl:value-of>. With this tag we can limit what we output more precisely.

<xsl:value-of> returns a value based on the expression in its attribute. The contents of the select= attribute could be an XML node or node-set, in which case the value returned might be the text content of the XML node or nodes. It could be the value of a variable (we’ll get to variables later, but they’re pretty much what you might think they are). It could be some static text or a number that we supply directly in the attribute. Or it could be the result of a complex expression, in which case what it returns could be a carefully tailored value. In all cases, though, <xsl:value-of> returns either a string or a number.

If we place static text in select=, it needs single quotes around it or we’ll get a syntax error.

Unlike <xsl:apply-templates>, <xsl:value-of> doesn’t kick off any additional processing. What you specify in select= is what you get, and it ends there. As we said, it’s a bit more precise than <xsl:apply-templates>.

To see an example in action, we’ll continue using the XML document in Figure 11, *Example XML for a cautionary note*, on page 3.

Suppose we have a couple of hundred products, each with a library of caution statements, and we want to make sure all the products have the correct caution statements. We want to know which products are mentioned in various caution statements. So all we need from the XML file is the value of the type= attribute and the text of any <product> tag.

When we match on the <caution>, we can get the text of the type= attribute with <xsl:value-of select="@type">. (The @ symbol before a node name means that the node is an attribute. To select an attribute, you always have to use @ before the attribute name.)

That just leaves the value of the <product> tag. We match on <paragraph> but this time we need to prevent the output of the <paragraph> text, since we don't want it. We do this by using the <xsl:apply-templates> tag with a select= statement that doesn't select for text. Instead, it selects only for the <product> tag, like so: <xsl:apply-templates select="product">. By defining what we want to select, we avoid invoking the default template for text.

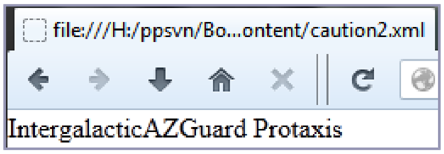
All we need now is a template for <product> that outputs its text (line 9), and we have what we wanted:

handlingcontent/caution2.xsl

```
Line 1  <?xml version="1.0" encoding="utf-8"?>
     -  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
     -    <xsl:template match="caution">
     -      <xsl:value-of select="@type"/>
     5      <xsl:apply-templates select="descendant::product"/>
     -    </xsl:template>
     -
     -    <xsl:template match="product">
     -      <xsl:value-of select="text()"/>
    10    </xsl:template>
     -  </xsl:stylesheet>
```

(We'll get more details about the text() function later in the book, but at the moment all we need to know is that it returns the text of a selected element —in this case, the current element.)

Change the XML file to point to the new caution2.xsl filename, then display the XML file in a browser again. The output now looks like this:



file:///H:/ppsvn/Bo...ontent/caution2.xml

```
IntergalacticAZGuard Protaxis
```

<xsl:value-of> has allowed us to fine tune the output, giving us only the parts we need.

Of course, to do that we had to step away from using the bare <xsl:apply-templates> instruction. We'll find in XSLT that we often have to trade off between giving explicit instructions and allowing the templates to work in a more functional, take-it-as-it-lays method. Our working methods will be determined by what we're trying to accomplish.

When outputting text, as we've done here, one thing to keep in mind: if the current content contains several text nodes that are separated by other nodes (such as elements or comments), <xsl:value-of> selects only the first node it finds. For example, in the following:

```
<p>Here is a point that <emph>must</emph> be emphasized!</p>
```

If our template matches on the <p> tag and contains <xsl:value-of select="text()">, our result will be "Here is a point that". So you'll need to exercise caution with this one.

You could also say <xsl:value-of select=".">, but like <xsl:apply-templates>, this approach gives the contents of all the elements contained. That dot in the select= attribute means to return the value of everything in the current context element—and that includes all of its children. That might even be what we want, but we need to be careful with using the dot.

<xsl:value-of> was just what we needed to get a specific slice of data out of our XML file. But you probably noticed that the output isn't exactly user friendly. The values of the type= attribute and <product> tag are run together, and we don't have any indication of what type of data each part represents.

It looks like we need to add some text to the output to indicate what the output is about. Fortunately, the next section shows a couple of ways to do that.