Locate specific sections of your XML documents with XPath, Part 1

Use XPath to navigate and select portions of an XML document

Skill Level: Intermediate

Brett D. McLaughlin, Sr. (brett@newInstance.com)
Author and Editor
O'Reilly Media, Inc.

10 Jun 2008

XML is a data format concerned primarily with compatibility and flexibility. But as useful as XML is, it's limited without the abilities to find specific portions of a document quickly and to filter and selectively locate data within a document. XPath provides the ability to easily reference specific text, elements, and attributes within a document—and with a fairly low learning curve. Additionally, XPath is key to many other XML vocabularies and technologies, such as XSL and XQuery. This tutorial will teach you the fundamentals of XPath, including all of its various selectors and semantics, in an example-driven and hands-on manner.

Section 1. Before you start

Learn what to expect from this tutorial and how to get the most out of it.

About this tutorial

Other tutorials in this series

- Part 2: Refine XPath results using predicate matching

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Part 1 of this tutorial details the XPath specification, which allows you to specify particular sections of an XML document using a directory-like syntax. You'll learn the syntax of XPath, and you'll work with tools that let you experiment with XPath. By the time you complete this tutorial, you'll be well beyond the basics of XPath. You'll have a solid understanding of nodes, wildcards, and how XPaths are evaluated, and you'll be able to combine the results of two different XPaths.

Objectives

Frequently used acronyms

- API: application programming interface
- HTML: Hypertext Markup Language
- URI: Uniform Resource Identifier
- W3C: World Wide Web Consortium
- XML: Extensible Markup Language
- XSL: Extensible Stylesheet Language
- XSLT: XSL Transformations

This tutorial takes you systematically through the foundational aspects of the XPath API, from its most basic syntax to its most common operators. It covers the various directory path navigations you can use, and it explains how those navigations affect the evaluation of your XPaths.

You'll also start to understand the larger context of how XPath relates to other XML-related specifications and technologies, such as XSL, XSLT, and XQuery. You'll not only be ready to talk intelligently about XPath in a job interview or programming round table, but you'll also be able to apply it to a variety of real-world problems.

Prerequisites

This tutorial is written for XML document authors and programmers. You should be familiar with and comfortable reading, writing, and manipulating XML. You should also be familiar with XML concepts, including the following:

- Elements
- Attributes
- Text

© Copyright IBM Corporation 1994, 2008. All rights reserved.

- The root element

Familiarity with the Document Object Model (DOM) is helpful for understanding *nodes*, but is not required. If you want to read up on the DOM, visit Resources to find several relevant links.

This tutorial will mention several other APIs and specifications as well, including XSL, XSLT, and XPath. Knowledge of any of these is helpful, but not required. For more information on any of these, consult Resources of the tutorial.

Section 2. Set up your environment for the examples

You'll work with an XML document throughout this tutorial. You'll need to have this document accessible on your machine, and you'll need to be familiar with the basic structure of the document. Additionally, you'll need a tool that executes your XPath locations and gives you feedback on what you've selected. This section explains how to get all of this working on your personal environment so you can follow along with the tutorial examples.

Unfortunately, the tools to evaluate XPath locations are still specific to each operating system. Some of the nicer tools that download as EXE files and run on Microsoft® Windows® won't work on Mac OS X. Similarly, the tools to work with XPath on Mac OS X won't run on Windows. While you can use Java™ programs and classes to create a system-independent means to work with XPath, this tutorial focuses on XPath rather than any particular programming language.

The following sections detail how to get a tool for working with XPath on both Windows and Mac OS X. Choose the section that you're interested in. Once you have a tool, all the syntax and examples shown throughout the rest of this tutorial will work on any platform; you'll just use your tool for evaluating the XPaths.

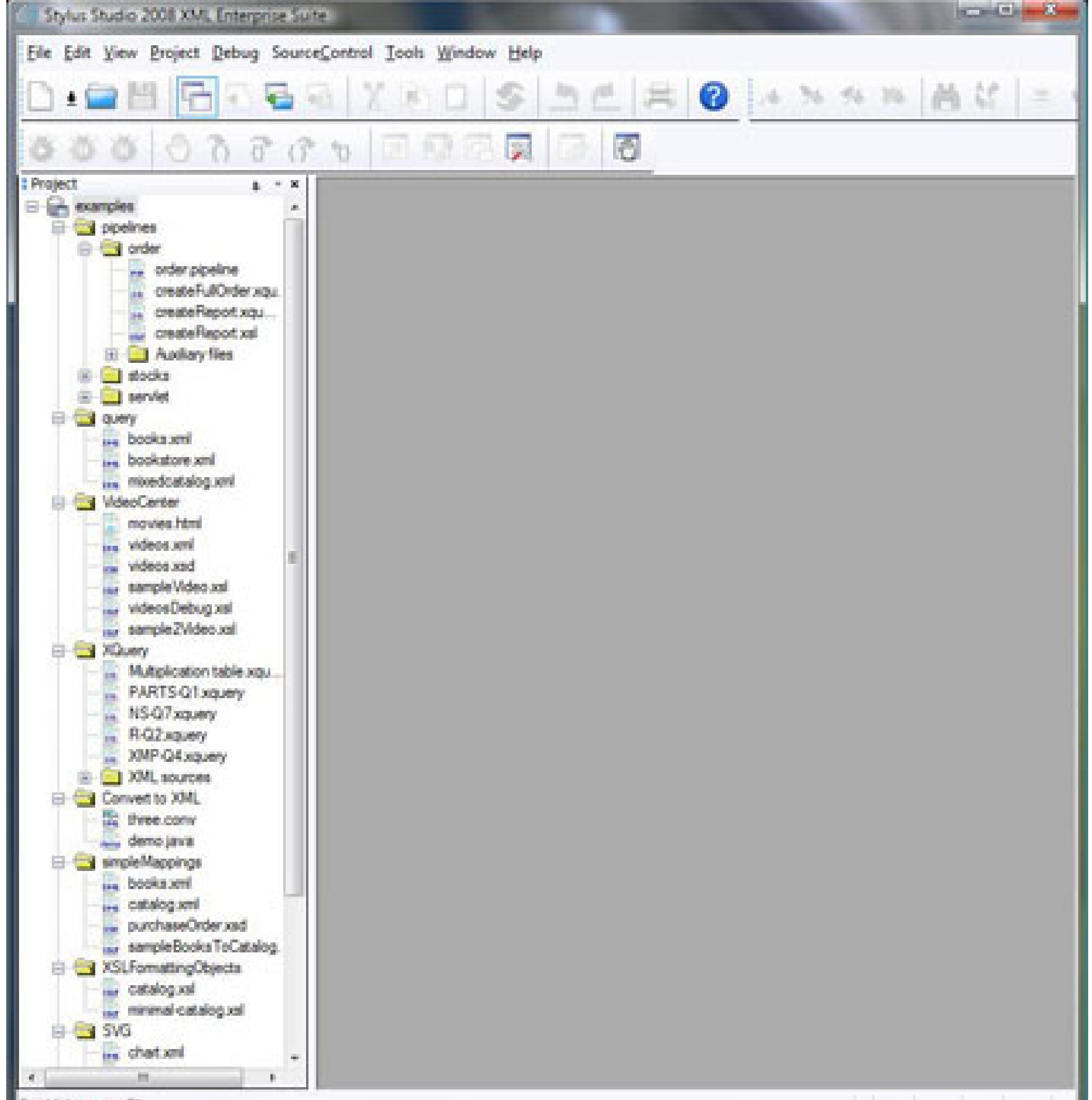
Evaluate XPaths on Windows

One of the better tools for working with XPath on Windows is Stylus Studio (see Resources for links to the Stylus Studio Web site and downloads page). Download any of the Stylus Studio trial versions—Enterprise Suite, Professional Suite, or Home Edition—and install them on your platform.

Once you install Stylus Studio, make sure you can open the program and load an XML document, and you're ready to work with XPaths. Your screen should look similar to Figure 1.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Figure 1. Stylus Studio allows you to evaluate XPath locations



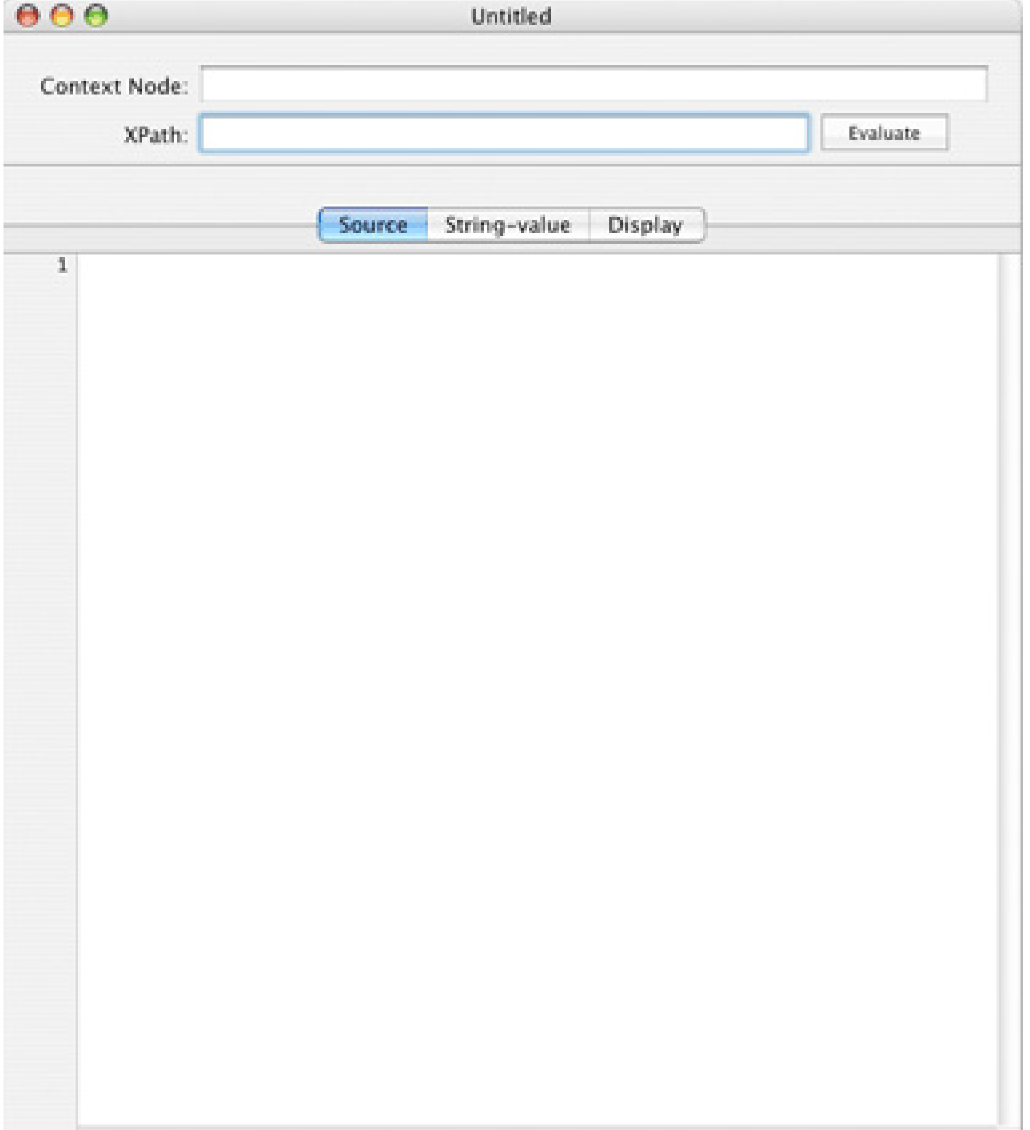
Evaluate XPaths on Mac OS X

The tools to work with XPath on Mac OS X—without using Java classes—are less refined and polished than their Windows counterparts. The most useful tool for learning XPath is AquaPath, which is open source and free for download (visit the download links in Resources). Download AquaPath as a disk image, and simply drag the AquaPath application from the mounted image into your Applications folder.

Double-click on the newly installed AquaPath application, and you should get a pretty spartan-looking screen, like the one in Figure 2.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Figure 2. AquaPath provides for XPath evaluation on Mac OS X



It doesn't look like much, but as you start to load XML documents and type in XPaths, this tool—like Stylus Studio—is worth its weight in gold.

An XML test document

XPath is more about XML than any other programming language. So while most programmers eventually move to using XPath through an API from Java or C# programming, this tutorial focuses purely on evaluating XPaths (through a tool such as AquaPath or Stylus Studio) against XML documents. It should go without saying, then, that you'll need an XML document to write XPaths against. And, for the sake of teaching, you'll need a document that's fairly lengthy (searching a 50-line document

© Copyright IBM Corporation 1994, 2008. All rights reserved.

isn't very impressive) and has lots of elements and attributes with data values.

Listing 1 shows just a portion of this XML document, which is the Ant build file from the Apache Xerces2 Java Parser. This document includes a lot more than is shown, but it would take pages just to display. However, you can download the entire XML document from Resources.

Listing 1. Sample XML document for tutorial

```
                    <?xml version="1.0"?>
<!--
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
-->
<!-- ===================================================================

Read the README file for build instruction.

Authors:
  Stefano Mazzocchi <stefano@apache.org>
  Anupam Bagchi     <abagchi@apache.org>
  Andy Clark, IBM

   $Id: build.xml 567790 2007-08-20 19:16:53Z mrglavas $

==================================================================== -->
<project default="usage" basedir=".">

  <!-- Xerces Java directories -->
  <property name="build.dir" value="./build"/>
  <property name="data.dir" value="./data"/>
  <property name="docs.dir" value="./docs"/>
  <property name="samples.dir" value="./samples"/>
  <property name="src.dir" value="./src"/>
  <property name="tests.dir" value="./tests"/>
  <property name="tools.dir" value="./tools"/>

  <!-- enable compilation under JDK 1.4 and above -->
  <taskdef name="xjavac" classname="org.apache.xerces.util.XJavac">
    <classpath>
      <pathelement location="${tools.dir}/bin/xjavac.jar"/>
    </classpath>
  </taskdef>

  <!-- Allow properties following these statements to be overridden -->
  <!-- Note that all of these don't have to exist.  They've just been defined
       in case they are used. -->
  <property file="build.properties"/>
  <property file=".ant.properties"/>
  <property file="${user.home}/.ant.properties"/>
  <property file="default.properties"/>

  <target name="init">
```

 © Copyright IBM Corporation 1994, 2008. All rights reserved.

```
    <property name='parser.Name' value='Xerces-J'/>
    <property name='parser.name' value='xerces-j'/>
    <property name='parser.shortname' value='xerces'/>
    <property name='parser.Version' value='2.9.1'/>
    <property name='parser.version' value='2.9.1'/>
    <property name='parser_version' value='2_9_1'/>

    <property name='deprecatedjar.parser' value='xerces.jar'/>
    <property name='jar.apis' value='xml-apis.jar'/>
    <property name='jar.parser' value='xercesImpl.jar'/>
    <property name='jar.samples' value='xercesSamples.jar'/>
    <property name='jar.dv' value='xercesDV.jar'/>
    <property name='jar.resolver' value='resolver.jar'/>
    <property name='jar.serializer' value='serializer.jar'/>
    <property name='jar.junit' value='junit.jar'/>

<!-- Lots more properties here... -->

  <!-- =================================================================== -->
  <!-- Prepares the build directory                                        -->
  <!-- =================================================================== -->
  <target name="prepare" depends="init">
    <mkdir dir="${build.dir}"/>
  </target>

  <!-- =================================================================== -->
  <!-- directory creation and file copying common to all configurations    -->
  <!-- =================================================================== -->
  <target name="prepare-common" depends="prepare">
    <!-- create directories -->
    <mkdir dir="${build.src}"/>
    <mkdir dir="${build.dest}"/>
    <mkdir dir="${build.dest}/META-INF"/>
    <mkdir dir="${build.dest}/META-INF/services"/>

    <copy
      file="${src.dir}/org/apache/xerces/jaxp/javax.xml.parsers.DocumentBuilderFactory"
      tofile="${build.dest}/META-INF/services/javax.xml.parsers.DocumentBuilderFactory"/>

    <copy
      file="${src.dir}/org/apache/xerces/jaxp/javax.xml.parsers.SAXParserFactory"
      tofile="${build.dest}/META-INF/services/javax.xml.parsers.SAXParserFactory"/>

    <copy
      file="${src.dir}/org/apache/xerces/jaxp/datatype/javax.xml.datatype.DatatypeFactory"
      tofile="${build.dest}/META-INF/services/javax.xml.datatype.DatatypeFactory"/>

    <copy file=
        "${src.dir}/org/apache/xerces/jaxp/validation/javax.xml.validation.SchemaFactory"
      tofile="${build.dest}/META-INF/services/javax.xml.validation.SchemaFactory"/>

    <copy file="${src.dir}/org/apache/xerces/parsers/org.xml.sax.driver"
      tofile="${build.dest}/META-INF/services/org.xml.sax.driver"/>

  </target>

<!-- Lots more targets and tasks here... -->

</project>
```

Make sure you have xerces-build.xml somewhere that's easily accessible, and that your XPath tool is installed and ready to use. With those two things in place, you're ready to start.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Section 3. Select elements

XPath is ultimately about one fundamental task: selecting nodes. I'll talk more about exactly what a node is in a bit, but for now, concentrate on the idea that XPath selects things. Those things can be elements, attributes, textual data—anything in your XML document. However, XPath isn't an API for doing much evaluation (it does what amounts to filtering, which is something I'll talk about in Part 2 of this series). XPath is really best at retrieving particular pieces of data.

Additionally, XML documents can be complex and long. Even the sample document for this tutorial, while relatively simple, is more than 1,400 lines long. When you deal with that much raw textual data, you need a way to locate specific sections of the document quickly, and that's what XPath is for. You can then use the programming language of your choice to work with, change, or delete that data.

Select the root node

The most fundamental of all components of an XML document— except perhaps the XML declaration, which is the first line of a document that begins with `<?xml` —is the document's root element. The *root element* is just the enclosing element of a document, within which all other elements, attributes, and text are contained.

It's important to realize that a root element is no different from any other element. It can have attributes, and it can contain other elements as well as textual data. There are no special rules about its naming or what it can contain. From an XPath perspective, selecting a root element is just like selecting any other element. The only difference for XPath is indicating where to start the search; in other words, the root element is the only element in a document that you *always* know the location of; it's the containing element, every time, no matter what.

To indicate the start of the search as the root element, then, your XPath location begins with a `/` character (the forward slash). For example, if you have a root element named `project`, the XPath in Listing 2 selects that element.

Listing 2. Select a root element named project

```
/project
```

Not only does this select a root element named `project`, but it also demonstrates two of the most basic parts of an XPath location:

- **Movement and hierarchical descriptors**: Syntax such as `/` or `..` that

© Copyright IBM Corporation 1994, 2008. All rights reserved.

navigate the XML hierarchy. Most of these look similar to the syntax you'd use to navigate around directories in a shell or command prompt window.

- **Node names**: Names such as "project" or "name" or "property" that refer to elements or attributes within the XML document. These refer to parts of the document at the current "level" that the movement descriptors place you at.

In Listing 2, the simplest of the movement descriptors is used: `/`. This instructs the XPath locator to begin at the root of the document. So to evaluate `/project`, the XPath evaluator begins its search at the root of the document. Then, at that level, it looks for an element named `project`.

To see this in action, try loading the xerces-build.xml document in your XPath evaluation tool, and enter `/project` in the XPath. You should see results similar to the screen in Figure 3.

Figure 3. Locate the root element

© Copyright IBM Corporation 1994, 2008. All rights reserved.

This visually demonstrates that you selected the root element. Note that it's not enough to just use the `/`, though; you've also need to get the root element's name right.

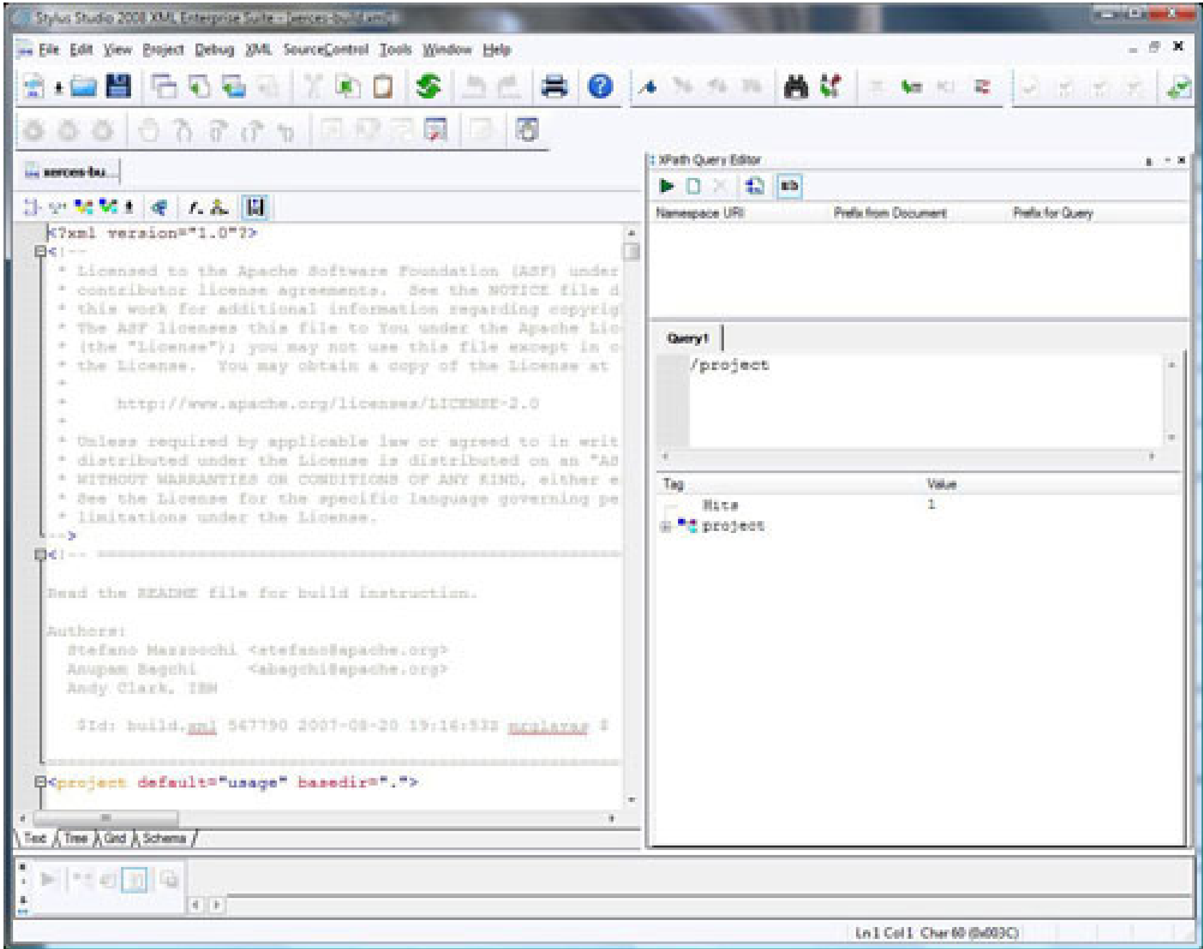
There are several more movement descriptors, which you'll learn about as this tutorial progresses.

Evaluate XPaths on Stylus Studio

Stylus Studio is a fairly high-end tool. Because of that, though, you'll have to do a little bit of work to get it to evaluate your XPath expressions. This won't trip you up, but it will take some knowledge about where everything is in the Stylus Studio application. First, start up Stylus Studio. Then, open up your XML source document; in this tutorial, that's the xerces-build.xml document you downloaded earlier. Along the top of your document window—which is *below* the icon strip for the entire application, just below a tab that should have your document name—are several icons. They're mostly odd symbols that are unclear in meaning. The rightmost of these is an "X" with an upward-pointing arrow between two black bars ( ). Click this, and a new pane opens along the right of your document. This is the XPath query editor, where you'll enter all your XPaths. Type your XPath into the second pane from the top, which is the one marked with a tab called Query1. Then, after you enter in your XPath, click the green arrow ( ) at the top of the XPath panel to evaluate your XPath (Stylus Studio calls this executing your query, in a very XQuery-centric sort of language). The results will appear in the bottom pane, as shown in Figure 4.

Figure 4. Locate the root element in Stylus Studio

© Copyright IBM Corporation 1994, 2008. All rights reserved.



This takes a little getting used to, but it's nice to have both the source document (in the main window) and the results of evaluating your XPath (in the bottom-right pane) next to each other. Additionally, you can expand the returned elements and see their children, attributes, and so forth. AquaPath on Mac OS X is a little more direct and simple in terms of the user interface, but it doesn't offer such a rich means of navigating through your results.

Select nested elements

Of course, selecting just the root element gets old quickly. That said, you'll spend most of your time selecting various elements when you work with XPath. You'll also most often work from the root element, because it allows you to set your starting location within the document at a known point.

Suppose you have elements named `property` nested within the root `project` element (which is exactly the case in the xerces-build.xml sample document). You already know how to select the root element; you just need to add to that the nested element named `property`. You that with an XPath like the one in Listing 3.

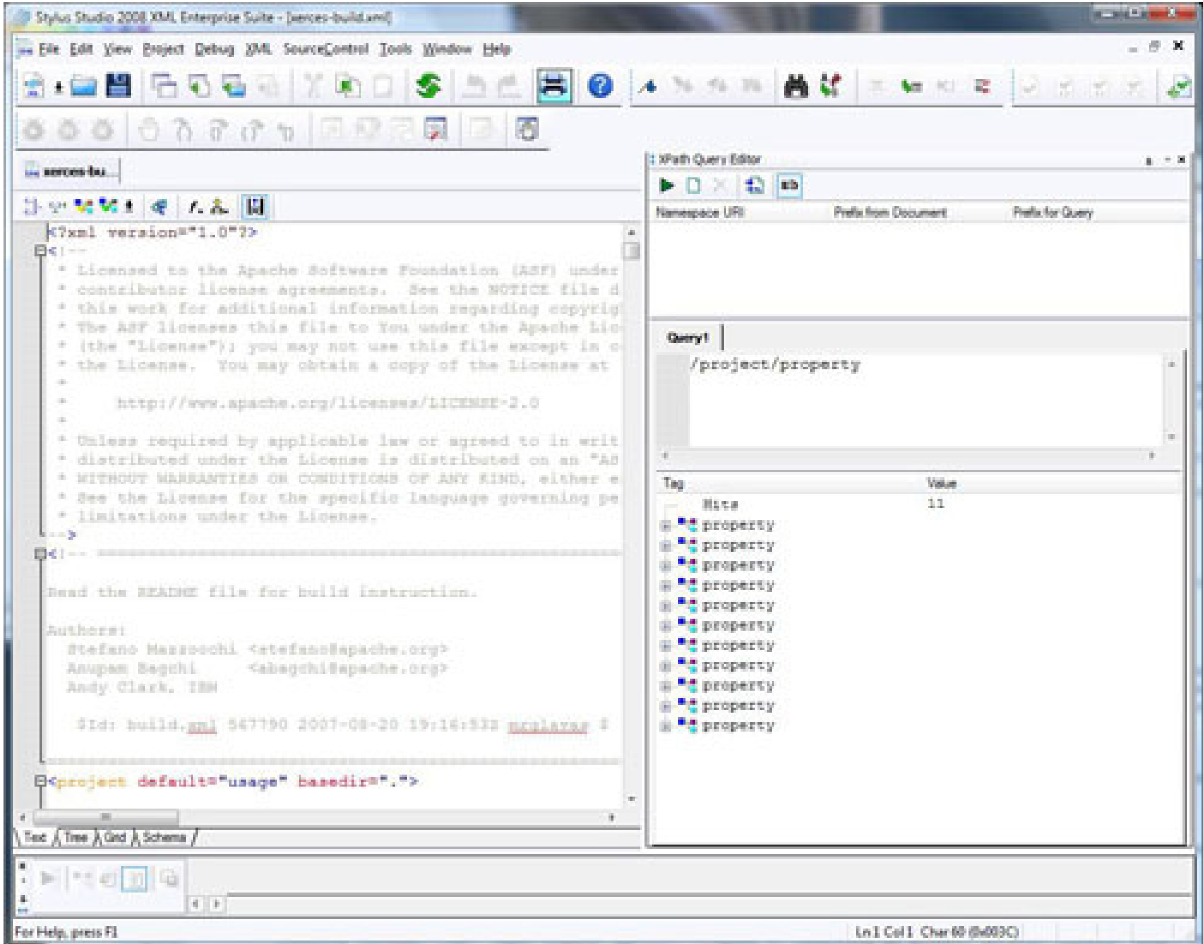
© Copyright IBM Corporation 1994, 2008. All rights reserved.

Listing 3. Select the property element directly underneath the root element

```
/project/property
```

Type this expression into your tool, and you should see something like Figure 5.

Figure 5. Select a nested element



> **AquaPath highlights, but doesn't focus**
>
> The AquaPath tool for Mac OS X highlights selected elements, attributes, and text when you tell it to evaluate an XPath expression. The selected nodes (more on what a node is shortly) appear in yellow. However, AquaPath does *not* move the focus of the screen to the first highlighted node or show only selected nodes. You only get however much of an XML document fits in the size of your AquaPath window. This makes it appear as if nothing is selected, and you'll have to do some scrolling to see just what was selected. This is definitely an inconvenience, so be prepared to work around it if you're on Mac OS X.

Again, the directory path metaphor should come to mind. Forward slashes (`/`)

© Copyright IBM Corporation 1994, 2008. All rights reserved.

separate elements from each other, and you can build up paths as deep as you want.

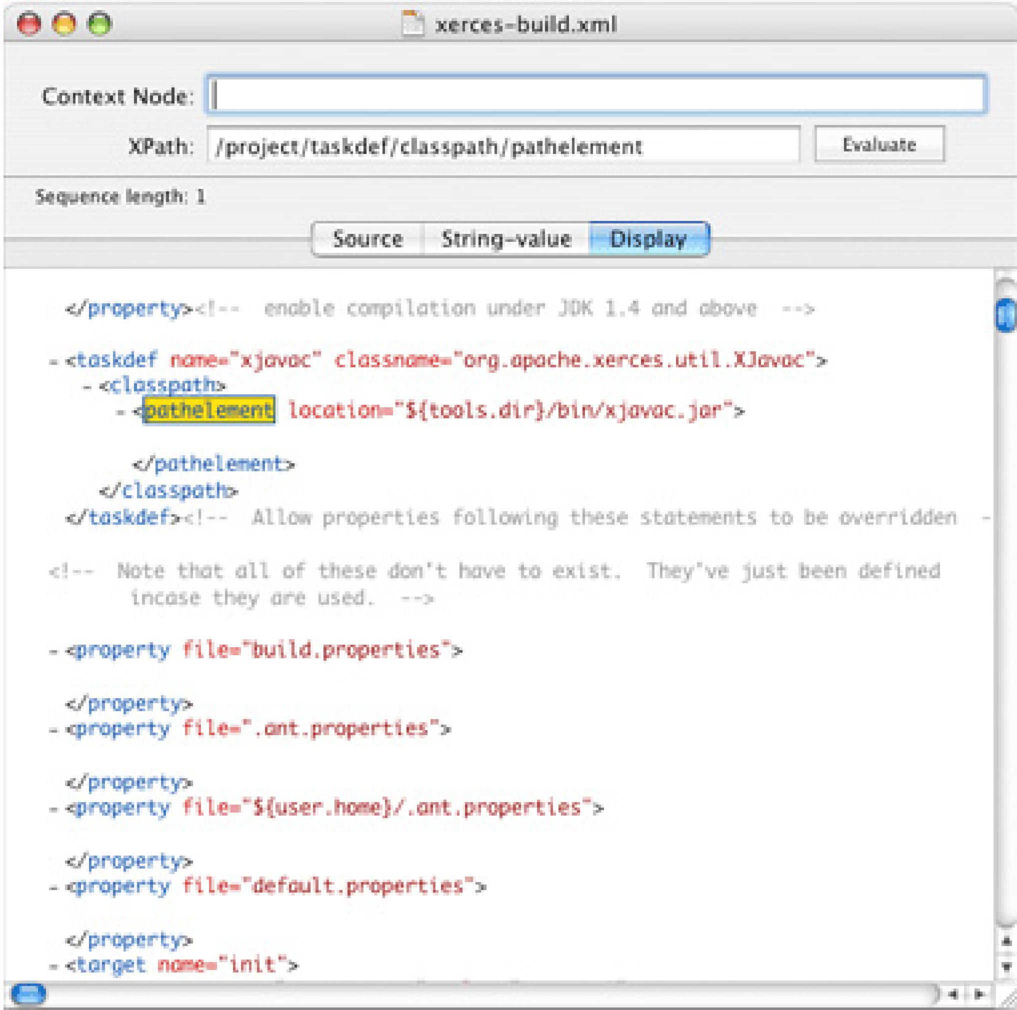
Listing 4 is another XPath that's slightly more complex but just as easy to understand.

Listing 4. Select the property element directly underneath the root element

```
/project/taskdef/classpath/pathelement
```

Figure 6 shows the elements selected by evaluating this XPath.

Figure 6. Select an element several levels deep



Before you move on to work with attributes, try to come up with some element-centric XPaths of your own. Here are a few things you can try to build XPaths to locate:

© Copyright IBM Corporation 1994, 2008. All rights reserved.

- All the `property` elements nested within `target` elements. `target` is nested directly under the root element (named `project`).
- All the `fileset` elements nested within `copy` elements. `copy` is usually nested within a `target` element, which is directly under the root element.
- All the `patternset` elements nested within `unzip` elements. `unzip` is also commonly nested within a `target` element, directly under the root element.
- All the `jar` elements nested within a `target` element.

Once you write all these XPaths, you're ready to move on to another type of selection: finding elements *anywhere* within a document.

Section 4. Select elements regardless of nesting

So far, you only looked at situations where you start from the root element of an XML document. There's another implication inherent with this, though: You must know the exact path to the elements you want to locate. For instance, the XPath `/project/taskdef/classpath/pathelement` is really only useful if two things are true:

1. You *know* that `pathelement` is nested within `classpath`, which is nested within `taskdef`, which is nested within `project`.
2. You *only* want `pathelement`s in that particular nesting, even if other `pathelement`s are in other parts of the XML document in a different nesting.

Select elements when you don't always have a specific nesting in mind

Lots of times, you know the constraint model that the document conforms to, and you want specific elements in specific nestings. But what if you want *all* `property` elements, no matter where they're located? Or perhaps you want to locate every `javac` element, but you *don't* have a good idea of the document's constraint model? In those cases, you need something more than a specific directory structure to match against.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Use a double slash for location-agnostic element matching

All the XPaths you've seen so far started with a forward slash, followed by the root element. That's because an initial slash moves to the root of a document, and then the first element is always the root element. However, you can use more than a single forward slash to start your XPath. Take a look at the XPath in Listing 5.

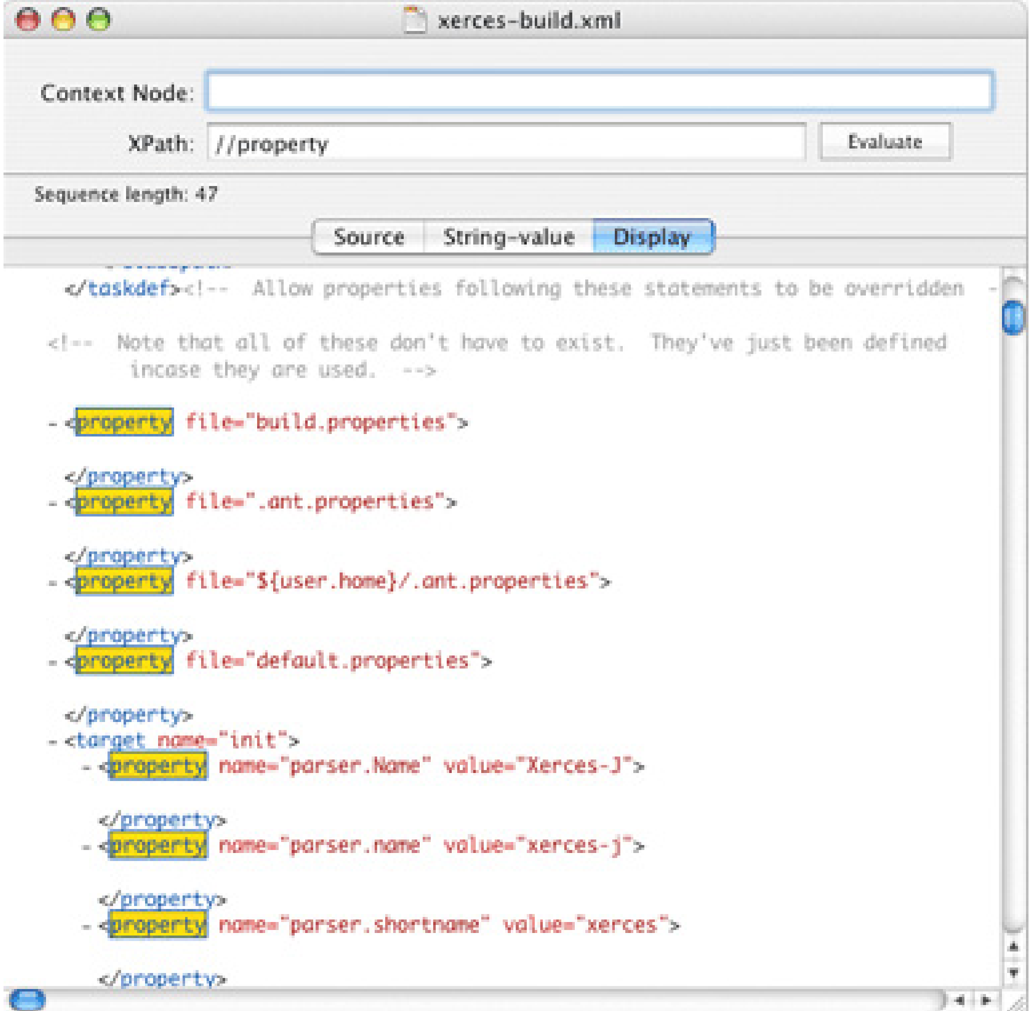
Listing 5. Start an XPath with a double slash

```
//property
```

This expression returns all `property` elements, *no matter where* they appear. So instead of starting at the root of a document, a double forward slash (`//`) indicates to an XPath evaluator to look anywhere in an XML document. Evaluate this expression, and your tool will return `property` elements nested within the root element, `project`, as well as within the `target` element with a `name` value of `init` (that's the first target, nearest the beginning of the document). You can see a sample results page in Figure 7.

Figure 7. Select an element regardless of location

© Copyright IBM Corporation 1994, 2008. All rights reserved.



Search for specific nestings anywhere in a document

You've learned two basic approaches to matching elements using XPath:

- Starting with the root element and specifying nestings using a single slash
- Indicating only the desired element name using a double slash

This seems pretty basic, but you can actually combine these two techniques to get some useful and complex results quickly. For instance, suppose you want to find `property` elements, but *only* if they're nested within `target` elements. So you don't want any `property` elements within the root `project` element.

This is a case where starting with the root element isn't helpful. First, you specifically *don't* want elements directly nested within the root element (something you haven't yet seen how to indicate to an XPath evaluator). Second, you don't care where the

© Copyright IBM Corporation 1994, 2008. All rights reserved.

`property` element is, as long as it's nested within `target`. So it seems obvious that you want to use the double slash in some form, because that's independent of a specific nesting.

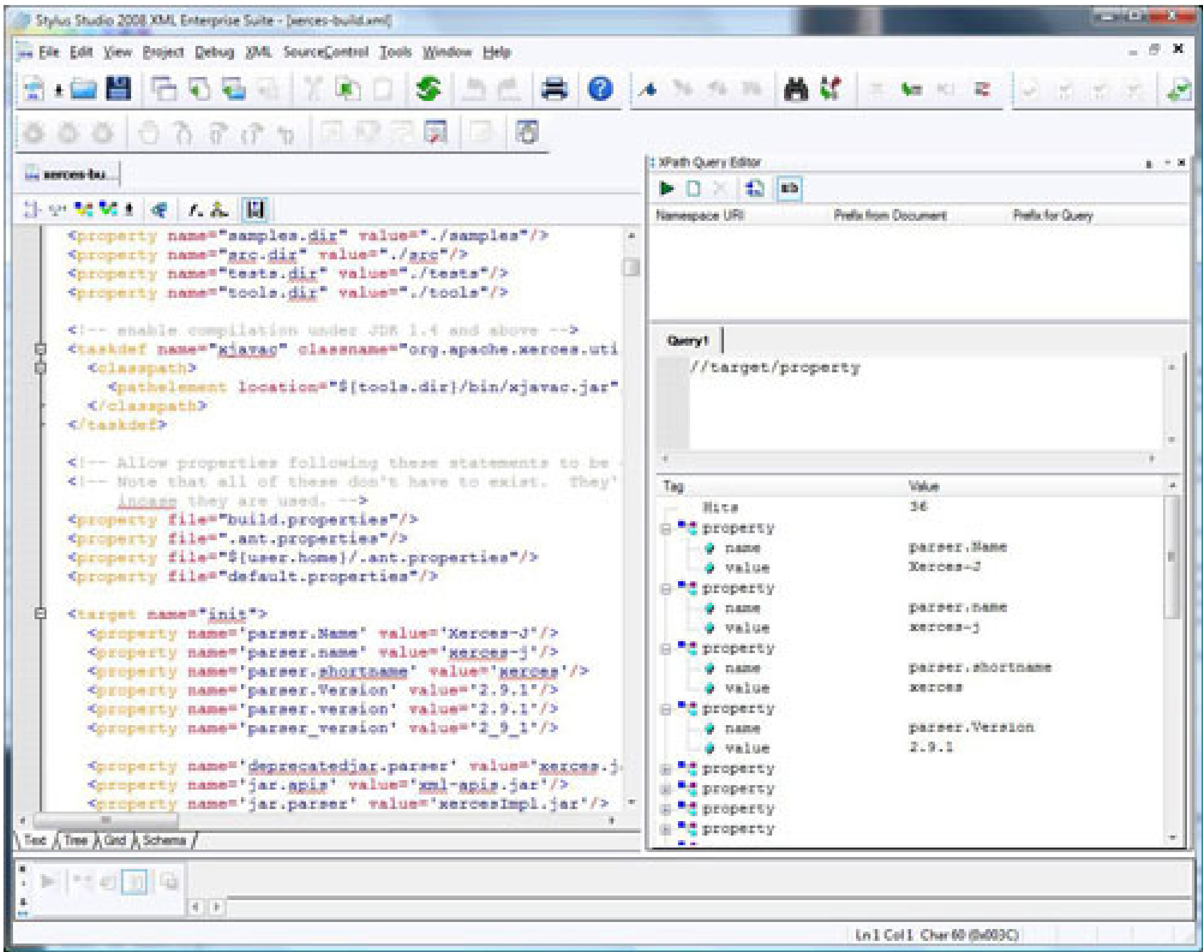
However, you do want to indicate one nesting: that `property` must be directly nested within `property`. Listing 6 is what you want.

Listing 6. Combine a double slash with a nesting

```
//target/property
```

Walk through this, and you'll see that it makes perfect sense. First, you want all `target` elements located anywhere in the document. The `//target` portion of the XPath gets you just that. However, instead of returning those elements, you want all `property` elements within those: `//target/property`. So you combine a specific nesting with the double slash. Evaluate this XPath expression, and you should get results similar to those in Figure 8.

Figure 8. Use double slash with a nesting



© Copyright IBM Corporation 1994, 2008. All rights reserved.

You can now write an XPath to locate the following items:

- All the `jar` elements anywhere in the XML document
- All the `attribute` elements nested within a `manifest` element that's nested within a `jar` element
- All `fileset` elements nested within `copy` elements

XPaths are evaluated piece by piece

Something you should have noticed above is the specific approach an XPath evaluator takes to processing your XPaths. Specifically, an XPath is evaluated piece by piece (it's actually node set by node set, but I'll get to that a bit later). First, the evaluator takes the first bit of syntax; so far, that's been either a single or double slash. That indicates whether to begin the search at the document root or anywhere within the document. Then, the evaluator finds all elements that match the element name (such as `project`) from whatever starting point was indicated.

At that point, you've basically located the evaluator at a specific place in the document. For example, for the partial XPath `/project`, anything after that part of the XPath is evaluated *relative to the current location*. That's why you can locate nested elements: The evaluator always evaluates one part of the expression and updates the location within the document. So `/project/task/property` first moves to the root, then to the `project` element, then to any nested `task` elements, and finally to any nested `property` elements.

Things get a little more complicated when you realize that an XPath expression can match multiple elements; in other words, the XPath `/project/task` matches *multiple* `task` elements nested within `/project`. Fortunately, the evaluator is smart enough to keep up with each location and continue further matches based on those locations.

Just to make things really interesting, consider an XPath like `//java/classpath`. Here, the evaluator begins by locating all `java` elements anywhere in the document—even at different levels of nesting. For each of those locations (all potentially disparate, and all potentially in totally different portions and nesting levels of the document), a nested `classpath` element is searched for, relative to that location. You should realize that the XPath evaluator is keeping up with lots of different locations, all for a pretty simple XPath.

For now, this doesn't affect you too much, but it soon will. The idea of an evaluator taking an expression piece by piece and updating its location within the document becomes critical when you start to deal with other constructs within a document—such as attributes.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Section 5. Select attributes

Anyone who's worked with XML much knows that attributes are just as important as elements. In fact, an XML API or technology that doesn't deal with attributes is more than half-broken; it's probably pretty close to useless. In the case of XPath, the primary goal of the specification (and tools like Stylus Studio or AquaPath that evaluate XPaths) is to select parts of an XML document. However, take a look at just a portion of the sample XML document that you've used:

```
    <property name='deprecatedjar.parser'
value='xerces.jar'/>
    <property name='jar.apis' value='xml-apis.jar'/>
    <property name='jar.parser'
value='xercesImpl.jar'/>
    <property name='jar.samples'
value='xercesSamples.jar'/>
    <property name='jar.dv' value='xercesDV.jar'/>
    <property name='jar.resolver'
value='resolver.jar'/>
    <property name='jar.serializer'
value='serializer.jar'/>
    <property name='jar.junit' value='junit.jar'/>
```

In this fragment, all the `property` elements are empty—in other words, each element has only attributes; it has no child elements or text. Without the ability to select the attributes on the `property` element, the document has little value. Of course, XPath is fully featured and can handle attribute selection with ease.

Select an attribute

To indicate an attribute, you simply preface the name of the attribute with the "at" sign (`@`) that you've used frequently in e-mail. For example, if you wanted to refer to an element named `name`, you'd use `@name`. By now, you should realize that XPath is all about location. Where in the document are you? What element are you located at? And in XML terms, what element are you expecting the attribute named `name` to appear on?

Apply what you already know

Before getting into details, you actually already have most of the knowledge you need to understand how to select attributes. Take a look at Listing 7, and take a (now educated) guess at what the XPath selects.

Listing 7. What attributes would this select?

© Copyright IBM Corporation 1994, 2008. All rights reserved.

```
//@name
```

You already know that the double slash tells an XPath evaluator to look anywhere in an XML document. You also know that `@name` selects attributes named `name`. So Listing 7 selects all attributes named `name` anywhere in the XML document (regardless of which element the attribute appears on).

Specify the element an attribute appears on

You've already seen that an XPath like `project/property` first selects the `project` element, then selects `property` elements nested within that. The slash separates the different parts of the path. In the same way, the XPath `property/@name` first selects elements named `property`, then (with the slash as the separator) selects attributes named `name` *on that element*. Put all this together, and you get an XPath like that in Listing 8.

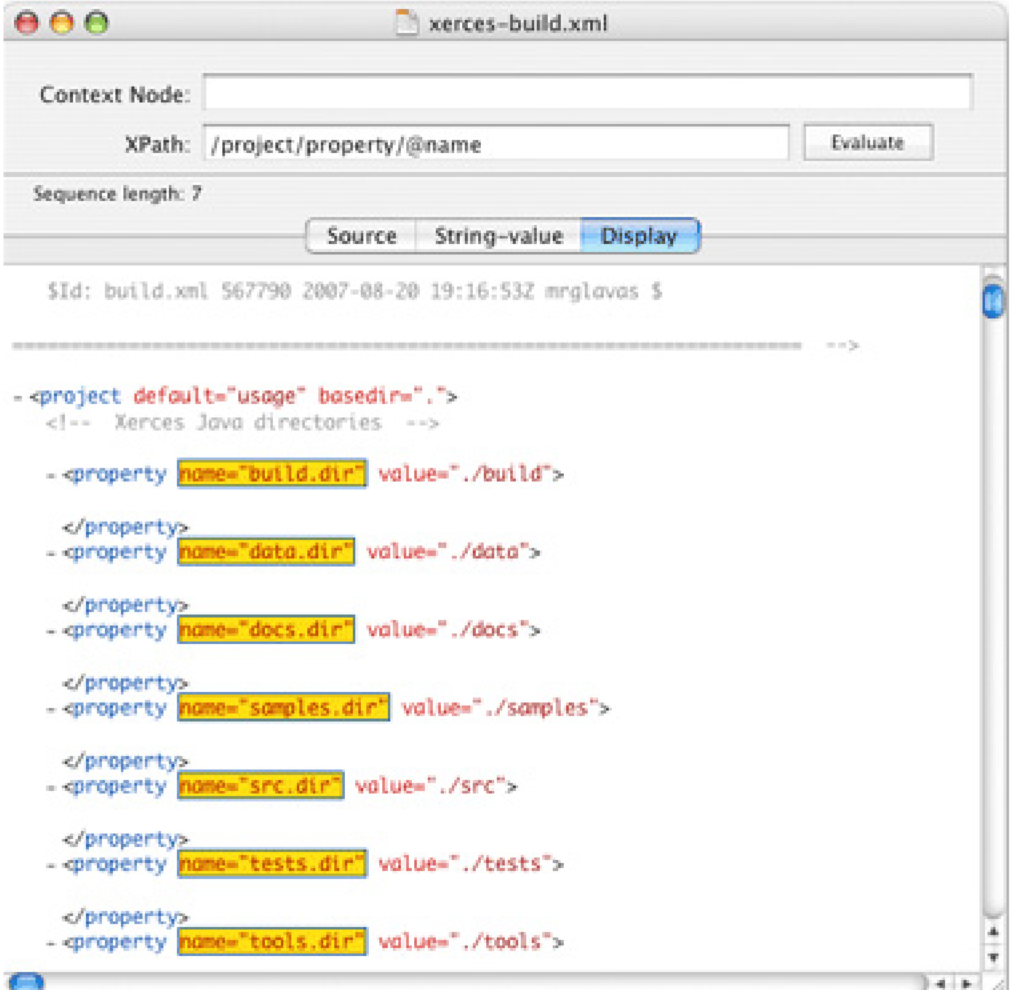
Listing 8. Select an attribute based on a complete path

```
/project/property/@name
```

Figure 9 shows the results of evaluating this XPath in AquaPath.

Figure 9. Select name attributes on specific property elements

© Copyright IBM Corporation 1994, 2008. All rights reserved.



Combine double slashes with attribute searches

Just as you used the double slash to widen your search with elements, you can do the same with attributes. Suppose you want to find *all* `name` attributes on `property` elements, regardless of whether the `property` element is nested within the root element (`project`). You can use an XPath like that shown in Listing 9.

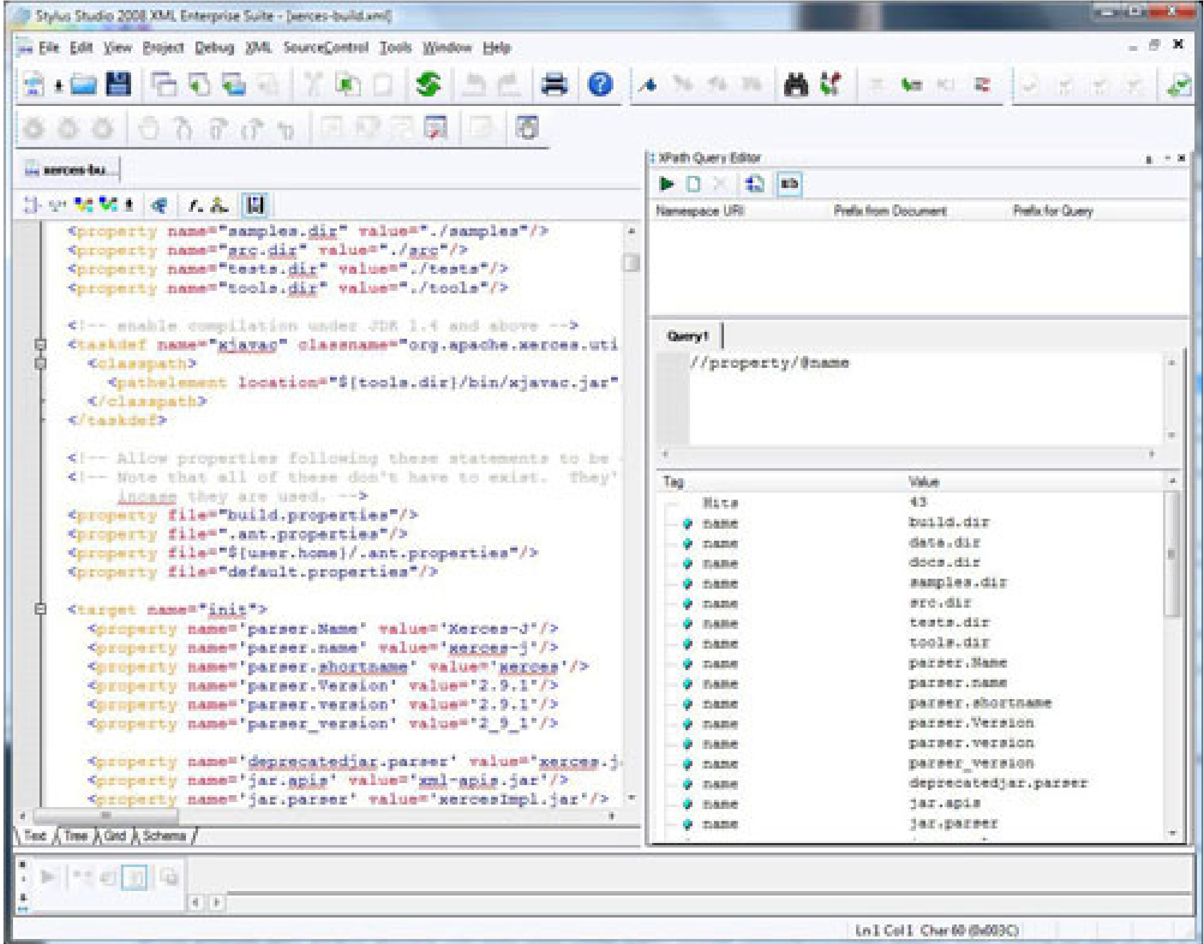
Listing 9. Widen the search for name attributes

```
//property/@name
```

This XPath, like the one in Listing 8, searches for `name` attributes on `property` elements, but this time the `property` element can appear anywhere within the document. Figure 10 shows the results of this less-constrained search.

Figure 10. Search for all property/@name attributes

© Copyright IBM Corporation 1994, 2008. All rights reserved.



Select the value of an attribute

In the case of elements, the goal (at least so far) has just been to select an element or an attribute. Before long, you'll want the *value* of a selected attribute. First, I should point out that this sort of evaluation starts to blur the line between XPath (a specification for selecting things) and XQuery (a specification that builds on XPath, does a lot more evaluation and complex querying, and is about returning values as much as elements and attributes).

Additionally, tools start to have more variance when you go beyond the fundamental XPaths discussed so far (and that most of this article focuses on). However, you can return the textual value of an attribute using a function called `string()`. Listing 10 is what you'd enter into your XPath evaluator.

Listing 10. Get the textual value of an attribute

```
string(/project/@default)
```


© Copyright IBM Corporation 1994, 2008. All rights reserved.

Try this out in AquaPath. You'll select the **String-value** tab to switch from selections of various parts of the document to a specific value, which is what the `string()` function returns. Your results should look like Figure 11.

Figure 11. Use the string() function in AquaPath

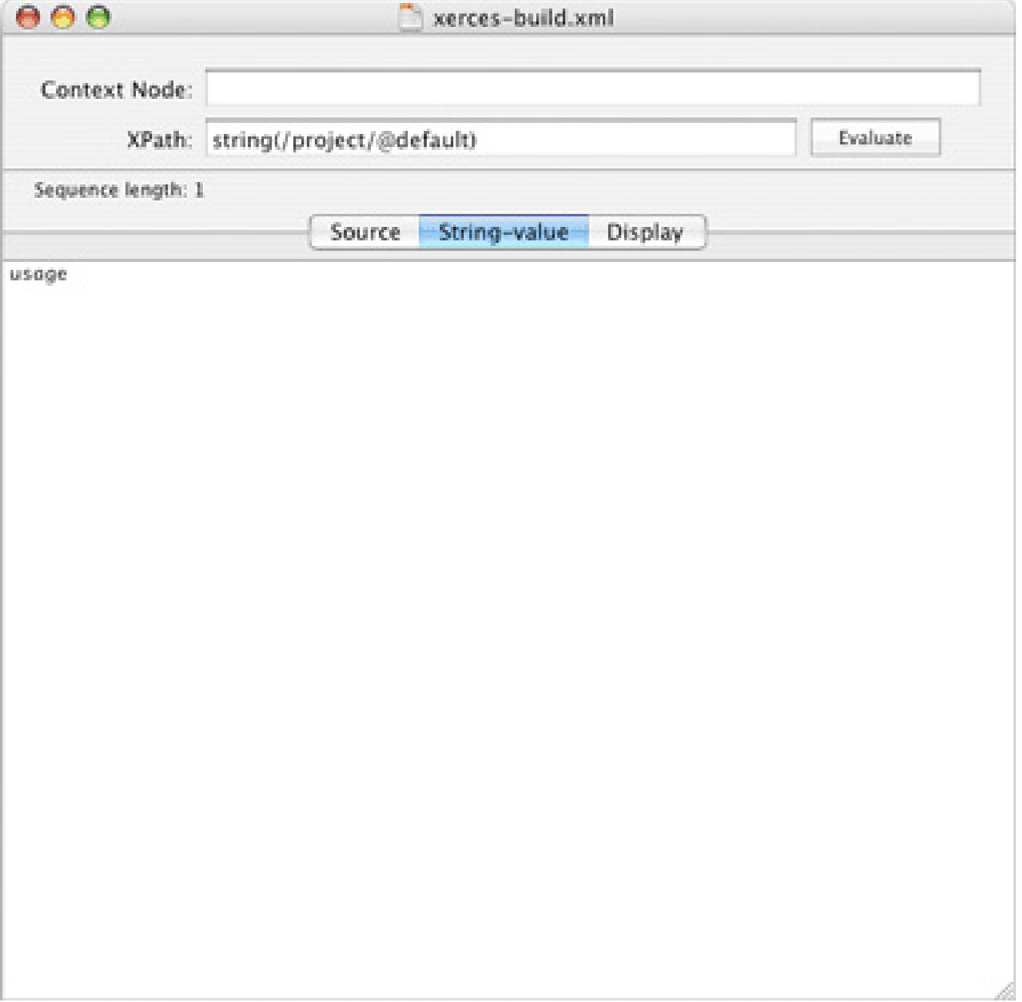
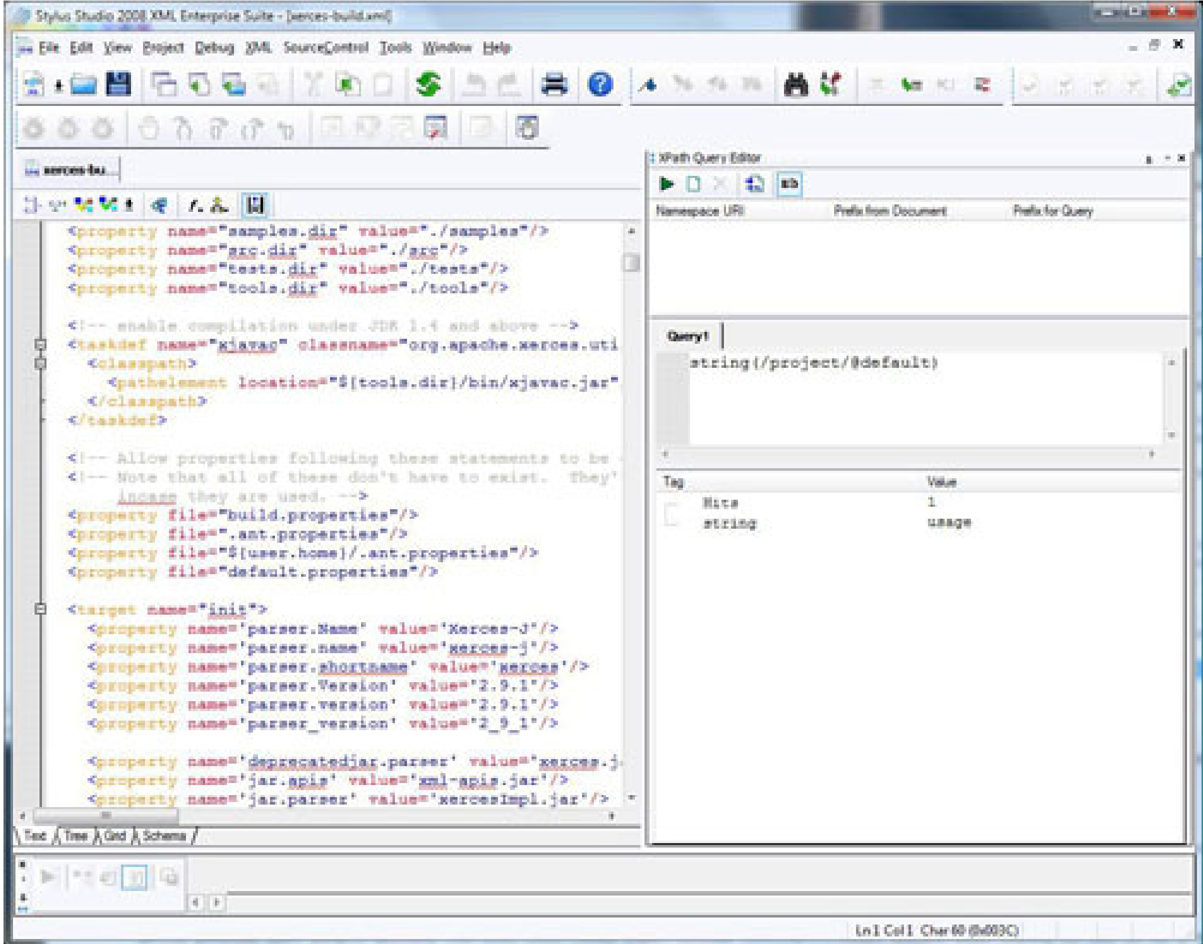


Figure 12 shows similar results in Stylus Studio.

Figure 12. Use the string() function in Stylus Studio

© Copyright IBM Corporation 1994, 2008. All rights reserved.



string() works on a singular attribute

The problem most people quickly run into with `string()` occurs when they try to use it on a set of values. For instance, take the XPath first shown in Listing 9: `//property/@name`. In the sample document, this XPath returns a number of attributes rather than just one. So trying to apply `string()` to this expression—like this: `string(//property/@name)` —generates an error. That's because `string()` takes a single attribute as an argument.

For this reason, unless you're particularly certain of your results, functions like `string()` are best saved for singular-valued attributes, such as an attribute on the root element of a document. By definition, a document can have only one root element, so you don't have to worry about multiple return values. That's why the XPath in Listing 10 works.

If you want to get into evaluating the value of multiple elements or attributes, you're moving further into the domain of XQuery (see Resources for links). For now, focus on selecting elements or attributes (or, soon, text), and worry about the values of the returned constructs a bit later.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Section 6. Widen the net with wildcards and the | operator

You've selected elements and attributes, and with the double slash, you're now able to search for those elements and attributes anywhere within a document. However, you're still limited by the specificity of your XPath. In other words, you need to know the names of the elements and attributes that you're looking for.

But, again, those limitations presuppose that you know exactly what you're looking for, even though you're no longer limited by *where* those elements and attributes appear. To move your XPath skills forward another level, you'll need to add wildcards to your toolbox, as well as the ability to *combine* two XPath locations to get the union of the results.

Use a wildcard to select all nodes

You're probably already ahead of the game with wildcards, because on almost all operating systems and programs, the `*` (asterisk) serves as a wildcard character. With XPath, the wildcard is a stand-in for all of a certain type. So, for example, a wildcard character can stand in for all elements at a certain position of your document, or all attributes—but not both.

Select all of an element's attributes

Consider the XPath you saw back in Listing 9: `//property/@name`. Now suppose you knew that you wanted to find all the `property` elements and then return all attributes from that element, regardless of their name. Maybe you don't know that `name` is an attribute, or you're concerned with all attributes (and their values). In that case, you can replace the attribute name (`name`) with a wildcard character, as in Listing 11.

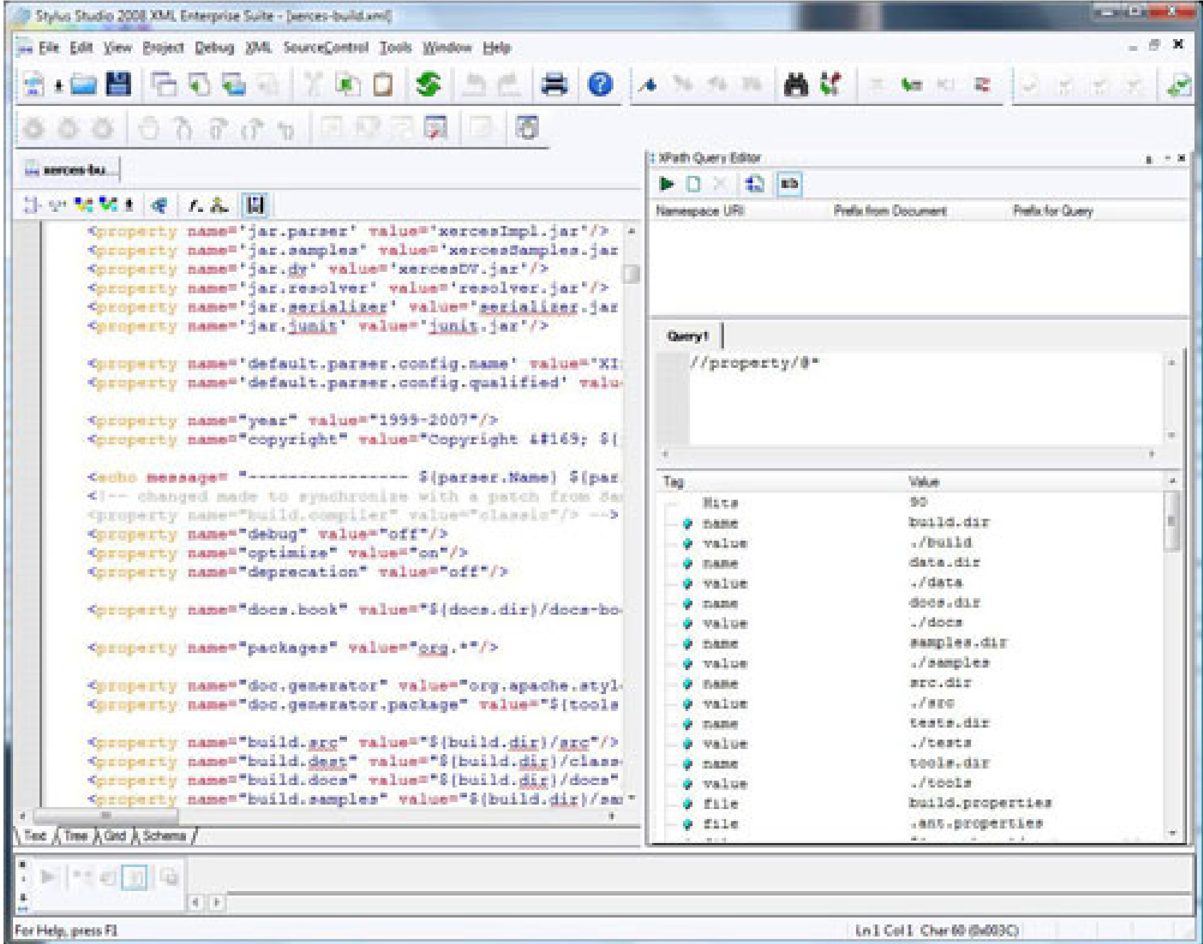
Listing 11. Use a wildcard to select all attributes on an element

```
//property/@*
```

Try this in your XPath evaluator, and you'll see something like Figure 13.

Figure 13. Retrieve all attributes on an element

© Copyright IBM Corporation 1994, 2008. All rights reserved.



The key here is to realize that the asterisk (`*`)simply *stands in* for an attribute name. So the XPath in Listing 11 takes on multiple values. In effect, it takes on `//property/@name`, `//property/@value`, `//property/@file`, and so on. In each case, the asterisk (`*`)> represents one single possible attribute name.

Use wildcards to select all child elements

In the same way, you can use wildcards to find all child elements of a particular element. Take the XPath `//target/copy/fileset`. You could easily widen that XPath with one like Listing 12.

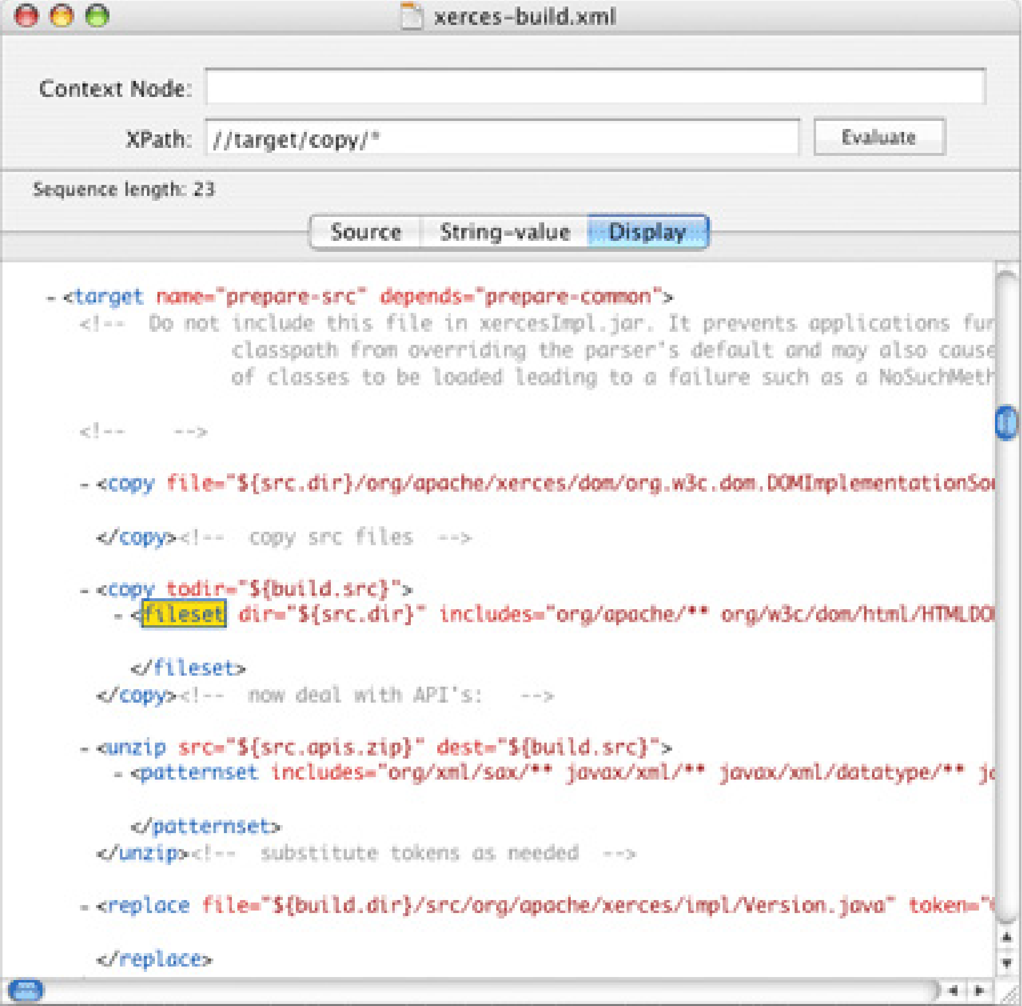
Listing 12. Use a wildcard to select all children of an element

```
//target/copy/*
```

This selects *all* child elements nested within `//target/copy`. So whether it's a `fileset` element or a `path` element, this expression will pick it up. Your results will vary based on your source document, but Figure 14 shows a portion of the result set on the example document you've been using.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Figure 14. Retrieve all child elements of an element



A wildcard always represents something

It's important to realize that the wildcard character stands in for *something* in the source document. That might seem like a vague and nebulous statement, but it's important. For instance, consider this fragment of XML:

```
<copy file=
       "${src.dir}/org/apache/xerces/jaxp/javax.xml.parsers.DocumentBuilderFactory"
      tofile=
       "${build.dest}/META-INF/services/javax.xml.parsers.DocumentBuilderFactory"/>

    <copy file=
       "${src.dir}/org/apache/xerces/jaxp/javax.xml.parsers.SAXParserFactory"
      tofile=
       "${build.dest}/META-INF/services/javax.xml.parsers.SAXParserFactory"/>
```

Both of these `copy` elements contain no nested child elements. So *neither* of these are returned by the XPath in Listing 12, `//target/copy/*`. Even though the first part of the XPath matches, the wildcard must represent something, and there's no

© Copyright IBM Corporation 1994, 2008. All rights reserved.

"something" (as in child elements) to it to represent here.

When you use the wildcard operator—on elements or attributes—realize that the wildcard must stand for something. The absence of value is *not* a valid replacement for the wildcard.

A wildcard always represents just one thing

In addition to the wildcard representing *something* in every case, it also represents *one* something. In other words, the wildcard in `//target/copy/*` represents just one level of nesting within the `copy` element. So if double nestings were present within `copy` —perhaps a `fileset` element with nested `path` elements within that—you'd only get the top-level nesting.

This becomes interesting when you consider that you can include a wildcard in the middle of an XPath expression, not just at the end. Listing 13 shows a useful example of the wildcard serving as the middle part of an XPath.

Listing 13. Use a wildcard in the middle of an XPath

```
//target/*/@name
```

This is more intuitive than you might think. Work the location through piece by piece, just as XPath evaluators would. First, the evaluator finds all `target` elements (the double slash indicates they can exist anywhere in the XML document you're searching). The wildcard tells the evaluator to find all child elements of `target`. So the element name doesn't matter, but any elements that are nested within a `target` are picked up. So far, this is nothing new. But then, the location continues: A slash (`/`) separates the wildcard from another part of the expression. Remember, XPath builds up locations, so at this stage, the evaluator has a bunch of child elements of `target` as the current location.

Then, the evaluator applies the partial XPath `@name`. That selects all attributes named `name` from the current element set. So elements named `property` or `taskdef` or anything else might have a `name` attribute, and all those matching attributes are returned.

This is one of the most useful ways to work with XPath and wildcards. Instead of just finding all the attributes of an element or all the children of an element, think about ways you can get what you want—perhaps a particular attribute or child element—without regard to the path you used to get to those attributes or children. You can even double up on wildcards; take a look at Listing 14.

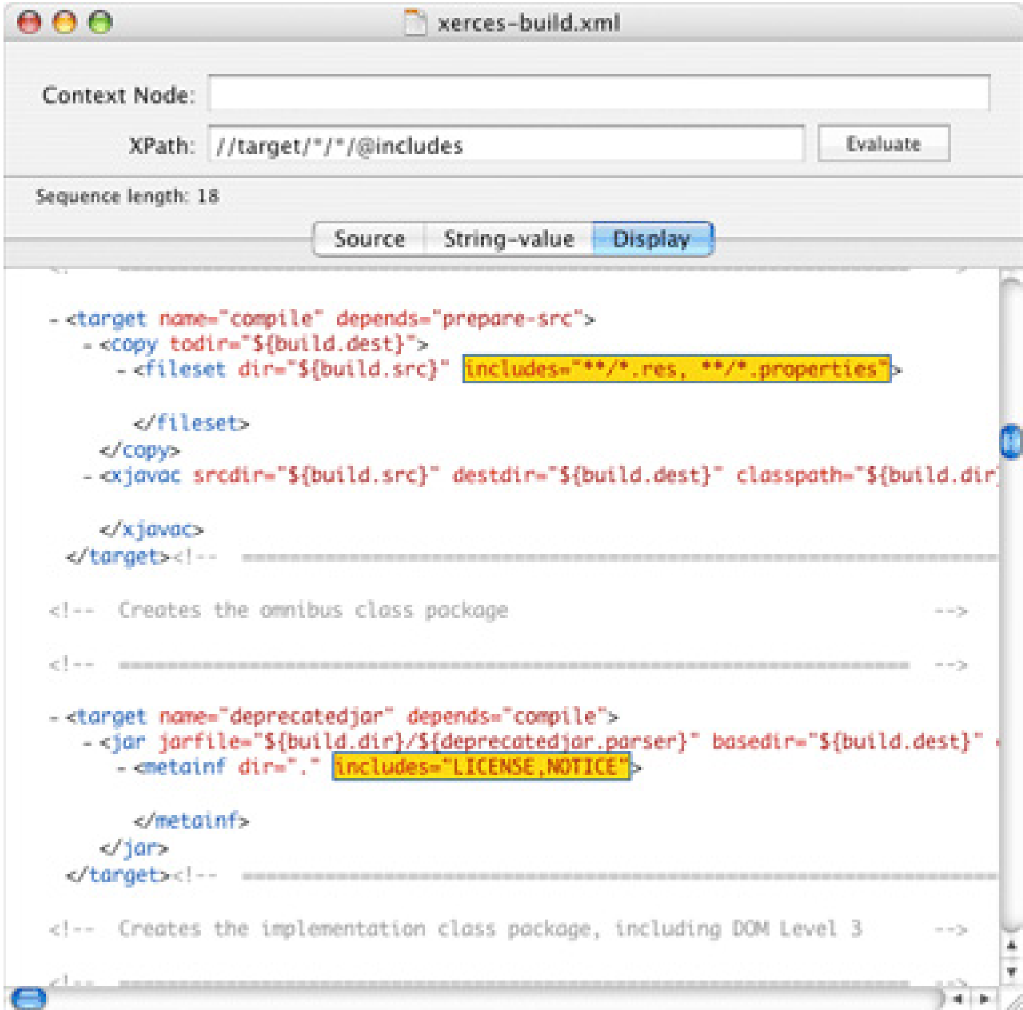
Listing 14. Use multiple wildcards in the same XPath

```
//target/*/*/@includes
```


© Copyright IBM Corporation 1994, 2008. All rights reserved.

Figure 15 shows the returned results from this XPath; the path to the `includes` attribute is pretty different in each case, but the level of nesting is the same in each case, as well as the ultimate destination: an attribute named `includes` that's nested two levels underneath a `target` element.

Figure 15. Use double wildcards in a single XPath



Combine XPath selections

The wildcard lets you substitute a character for a particular part of an XPath. Still, sometimes you'll want all of *this*, in addition to all of *that*. In other words, you'll want to combine the results of two XPaths. That's where the OR character comes in, which is represented syntactically by the pipe symbol: `|`.

Simply put, you use the OR symbol between two XPaths, and the result is a new XPath. Listing 15 shows a simple example.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Listing 15. Combine two XPaths

```
/project/property | /project/task/property
```

In this case, two different locations are combined into a single result set. All the property elements under the root are returned, *as well as* all the property elements under task elements under the root. This is functionally different from an XPath you've already seen, //property, because if a property element was nested several layers deep or under an element other than task or the root, it wouldn't be returned by the XPath in Listing 15. (It's worth noting that while //property is different than the XPath in Listing 15, the result set is the same in the example XML document, so they're the same in terms of practical use.)

Select disparate elements and attributes

This sort of combination gets much more interesting when you expand from simply selecting the same element in different locations to selecting different elements. For instance, consider the XPath in Listing 16.

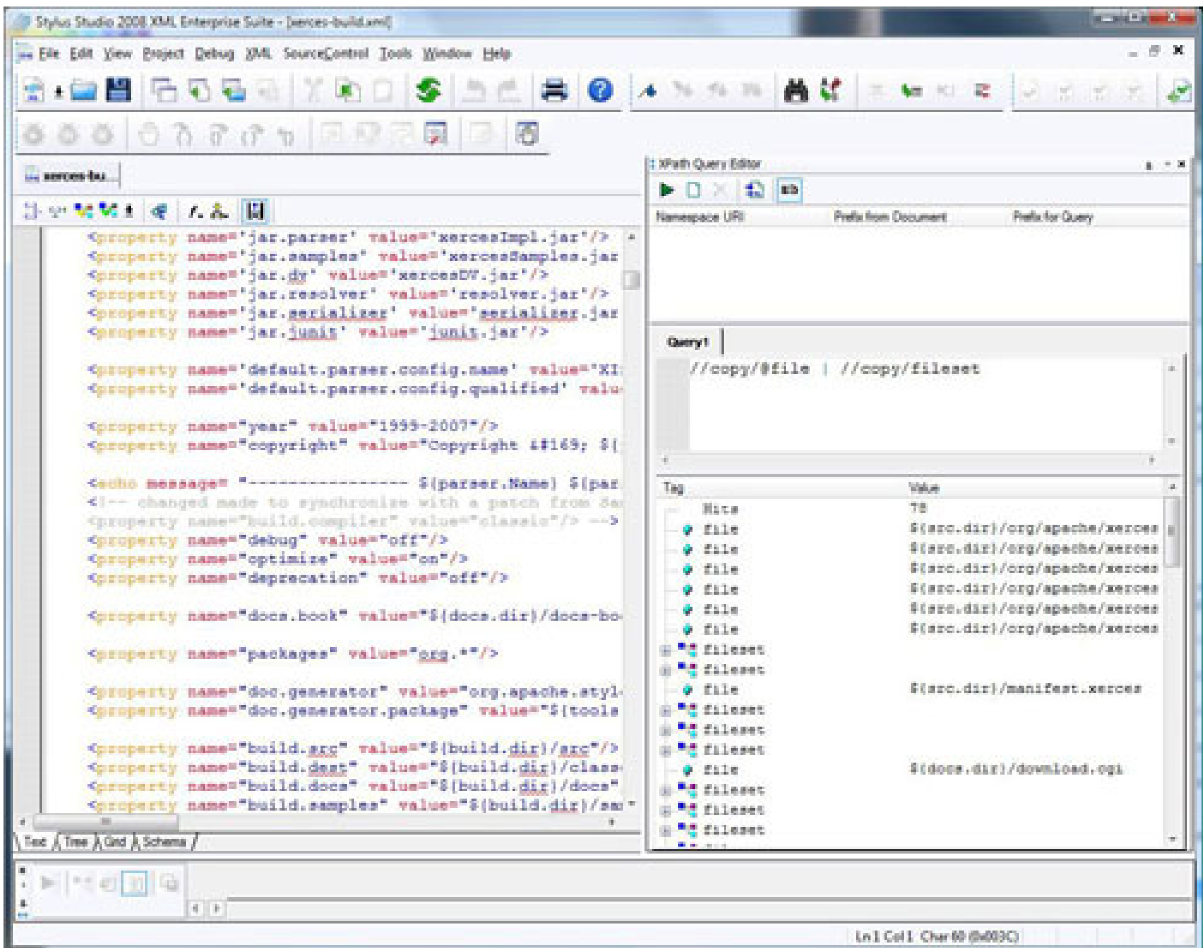
Listing 16. Combine multiple XPaths

```
//copy/@file | //copy/fileset
```

Figure 16 shows the results of evaluating this XPath.

Figure 16. Search for two XPaths at once

© Copyright IBM Corporation 1994, 2008. All rights reserved.



This combines a search for `file` attributes on `copy` elements with child elements of `copy` named `fileset`. In essence, it gets all the files that might be copied, no matter how they're represented in the build file.

Of course, you can get as complex as you want. You can include wildcards, and you can join more than just two XPaths. It's all up to you. Listing 17 contains a few ideas for you to experiment with. These XPaths should hint at just what's possible using wildcards and the pipe operator.

Listing 17. A few interesting XPaths using wildcards and pipes

```
                    //target/@name | //target/@depends
//copy/@todir | //copy/fileset/@dir
//gzip/@zipfile | //tar/@tarfile | //delete/@file
//target/*/@dir | //target/*/fileset
```

AND or OR?

Before leaving wildcards and the pipe (|) character, I want ot deal with a conceptual issue: The naming of the pipe operator is a bit confusing. In many cases, you'll see it

© Copyright IBM Corporation 1994, 2008. All rights reserved.

referred to as an OR, although that flies in the face of a lot of programming language semantics. OR generally applies to boolean statements, where either one condition OR the other is true. That's somewhat similar to what happens in XPath, but not quite a direct match.

You'll also see the pipe called an AND, which is meant to suggest "the results in this XPath AND the results in that XPath." That's also a bit counterintuitive, though. Usually, AND implies that both clauses must be true; however, in using the pipe (|), you really say that if *either* XPath applies, include the matching node.

Using the pipe in XPath is really a *union* operator, not an AND or an OR. The results of the first XPath are combined with the results of the second XPath. If you consider that an XPath really returns a set of elements or attributes (or text), then you combine those sets. That's a union operation, even though you'll rarely see XPath documentation refer to unions. Whatever the case, it's best to simply think about using the pipe (|) rather than getting hung up on AND and OR.

Section 7. What exactly are nodes?

Here's a term you've seen several times now, with no concrete definition: *node*. While you've already got a lot of XPath under your fingers, it's really a firm understanding of nodes that will take you from XPath user to XPath guru.

Nodes are generic

A node, at its simplest, is a vague, undefined data type that serves as a placeholder for specifications like XPath and APIs like DOM. If you're familiar with Java, C#, or another object-oriented programming language, a node is akin to an interface, or perhaps an abstract class. All the types of XML constructs that you've selected—elements, attributes, and text—are nodes. So there are element nodes, attribute nodes, and text nodes. (Actually, there are more, but those are the three primary types you need to know about to use XPath well.)

Nodes exist so that an XPath evaluation tool has some basic data type it can return for all XPaths. Rather than worry about an XPath that returns elements, or an XPath that returns attributes, and then distinguish between the two, all XPaths return nodes. In most cases, the nodes returned are all of the same type. But when you start combining XPaths together, you might have attributes and elements returned from a compound XPath. None of that matters, though, because ultimately, the complete XPath just returns nodes.

 © Copyright IBM Corporation 1994, 2008. All rights reserved.

Of course, your brain, along with the brain of every other document author and programmer who works with XML, is wired to think in elements and attributes. That's okay; nothing is wrong with that. Just realize that at a more fundamental level, an XPath returns nodes. Just as a `Dog` or `Cat` instance might inherit from a more basic `Animal` class, elements and attributes are all ultimately nodes, at least in XPath parlance.

XPath deals in sets of nodes

Not only do all XPaths return nodes, they always return a *set* of nodes. If you took any linear algebra in high school or college, you'll remember that a set is just a collection. Further, a set can have zero, one, or more members. So in the XPath realm, an XPath might return zero nodes, a single node, or more than one node. For instance, the XPath `//target` returns 39 nodes in the example document; the XPath `//gzip/@zipfile | //tar/@tarfile | //delete/@file` returns 14.

Inherent with the idea of a node set—and the fact that a node set can have zero members—is the difference between an invalid XPath and one that's valid but returns no members. For instance, an invalid XPath has syntax that's illegal or misused, such as `//gzip/@zipfile || //tar/@tarfile`. A double pipe is used here instead of a single pipe, making the XPath invalid. Tools like AquaPath and Stylus Studio report this as an error or invalid expression. On the other hand, the XPath `//target/*/*/*/@dir` is valid, but it returns no nodes (an empty node set, but a node set nonetheless).

When you start to think about each XPath returning a set of nodes, the pipe (`|`) operator also makes more sense. It functions more like a true union operation. Each individual XPath is evaluated, and multiple node sets are returned; then those sets are combined into one larger node set. That's also how you can end up with a set of nodes that aren't all the same; you can have elements, attributes, and even text.

From a purely syntactical point of view, understanding nodes isn't going to change the way you write your XPaths. However, it should change the way you think about them. It should also help you distinguish between an empty node set—from a valid XPath—and an error caused by an invalid XPath.

Section 8. Conclusion

As Part 1 of this tutorial on XPath winds down, you should feel like you have a real head start on using XPath, and really understanding its fundamental operations and a lot of its operators. Just to wrap up, I'll make sure the key points are in the forefront

© Copyright IBM Corporation 1994, 2008. All rights reserved.

of your brain.

Takeaways

First and foremost, XPath is about selection. XSL and XSLT, XQuery, and APIs like Java API for XML Processing (JAXP) and TrAX for Java are all for processing XML, and many of them build upon XPath. However, XPath is always going to be about selecting something (or multiple somethings). It's then up to you—the document author or application programmer or XML manipulator—to work with that selection. As long as you don't expect XPath to do fancy calculations or perform any sort of iterative calculations on your attributes, you're off to a good start.

Next, XPath is a node-based specification. XPath doesn't think about things in terms of elements and attributes, but rather in terms of nodes (which have types, but are ultimately simply nodes). When you come up with an XPath, you're returning a set of nodes: zero, one, or more. Often, you'll end up with a set of nodes that are of the same type (that's almost always the case unless you're doing a union of more than one XPath), but XPath still thinks of things in terms of nodes.

Finally, remember that XPaths are evaluated piece by piece. First, the initial slash or double slash is dealt with (if one or the other is present), then the element or wildcard, and so on. Wildcards (`*`s) stand in for just one piece of an XPath, and you always need to update your mental location as you figure out how to evaluate an XPath.

Notice that none of these three key takeaways are syntactical in nature. Syntax, ultimately, is just a Google search, a reference book, or a developerWorks tutorial away. The concepts involved, though, will take you well beyond where rote memorization will.

What's next?

Other tutorials in this series

- Part 2: Refine XPath results using predicate matching

In Part 2 of this tutorial, you'll push beyond pure selection based on names, positioning, and the relationship between elements. You'll learn about *predicates* and how there's still an entirely new world of selection awaiting you. With predicates, it's possible to add criteria to your matching. Suppose you want only the `target` elements with a name of `init`, or you only want `property` elements that have a `value` attribute and are children of the root element. The key to these increasingly complicated XPaths is predicate matching.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

While you take a break and get ready to tackle Part 2, play around with your XPath evaluation tools. AquaPath and Stylus Studio are great for really understanding what XPaths evaluate to and *how* those evaluations occur. The visual interface also makes it easy to play around, which is something you should do before you move on to predicates.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Downloads

Description	Name	Size	Download method
Example source code	xpath-2_exampleXml.zip	57KB	HTTP

Information about download methods

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Resources

Learn

- Locate specific sections of your XML documents with XPath, Part 2: Refine XPath results using predicate matching (Brett McLaughlin, developerWorks, June 2008): Add predicates to your XPath skills and find the exact nodes you want when you evaluate attribute values plus the parent and child nodes of a target element.
- XPath 1.0: Read the formal definition of XPath in the original specification.
- XPath 2.0: Read the online specification for the most current version of XPath.
- Tutorial on XPath: Understand how XPath is fundamental to much advanced XML usage with this useful but brief tutorial.
- Document Object Model (DOM): Explore the concept of nodes, used extensively in XPath, more fully in this article on the W3C site.
- Understanding DOM (Nicholas Chase (developerWorks, March 2007): Dig deeper into manipulating XML from a node-based API in an excellent tutorial.
- Stylus Studio: Learn and use a robust set of XML-related tools, including an XPath evaluator. Numerous white papers on XML are also available from the Stylus Studio Web site.
- Apache Xerces2 Java Parser: The sample XML document used in this tutorial is a build file from this XML parser. Get the file from Download.
- IBM XML certification: Find out how you can become an IBM-Certified Developer in XML and related technologies.
- XML technical library: See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- developerWorks technical events and webcasts: Stay current with technology in these sessions.
- The technology bookstore: Browse for books on these and other technical topics.
- developerWorks podcasts: Listen to interesting interviews and discussions for software developers.

Get products and technologies

- Stylus Studio 2008 XML: Download to get started with XPath and XML documents on the Windows platform.
- AquaPath: Download to enable easy XPath location evaluation on Mac OS X.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

- Scandalous Software: Check out a solid set of XML-related tools, all targeted at the Mac OS X platform.
- Java & XML, Third Edition (Brett McLaughlin and Justin Edelson, O'Reilly Media, 2006): Cover XML from start to finish, including extensive information on various XML vocabularies.
- IBM trial software for product evaluation: Build your next project with trial software available for download directly from developerWorks, including application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Participate in the discussion forum for this content.
- XML zone discussion forums: Participate in any of several XML-related discussions.
- developerWorks XML zone: Share your thoughts: After you read this article, post your comments and thoughts in this forum. The XML zone editors moderate the forum and welcome your input.
- developerWorks blogs: Check out these blogs and get involved in the developerWorks community.

About the author

Brett D. McLaughlin, Sr.
Brett McLaughlin is a bestselling and award-winning non-fiction author. His books on computer programming, home theater, and analysis and design have sold in excess of 100,000 copies. He has been writing, editing, and producing technical books for nearly a decade, and is as comfortable in front of a word processor as he is behind a guitar, chasing his two sons around the house, or laughing at reruns of Arrested Development with his wife. His last book, Head First Object Oriented Analysis and Design, won the 2007 Jolt Technical Book award. His classic Java and XML remains one of the definitive works on using XML technologies in the Java language.

Trademarks

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli,

© Copyright IBM Corporation 1994, 2008. All rights reserved.

WebSphere, and pureXML are trademarks of IBM Corporation in the United States, other countries, or both.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Other company, product, or service names may be trademarks or service marks of others.

© Copyright IBM Corporation 1994, 2008. All rights reserved.