IBM

developerWorks

Process Atom 1.0 with XSLT

Use XPath expressions to navigate Atom documents, create or transform Atom source files using XSLT

Skill Level: Intermediate

Uche Ogbuji (uche.ogbuji@fourthought.com)
Principal Consultant
Fourthought, Inc.

13 Dec 2005

Atom 1.0 is the emerging Internet Engineering Task Force (IETF) standard for Web feeds -- information updates on Web site contents. Since Atom is an XML format, XSLT is a powerful tool for processing it. In this tutorial, Uche Ogbuji looks at XSLT techniques for processing Atom documents, addressing real-life use cases.

Section 1. Before you start

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

About this tutorial

This tutorial provides a step-by-step look at how to process Atom 1.0 with XSLT. Atom is an important format for conveying information about Web sites that are organized into episodic content in some way. This might be Weblogs where new entries are published from time to time, event listings, multi-media programs and schedules, and much more. The breadth of Atom's domain is illustrated in the following description on the home page:

> Atom is the name of an XML-based Web content and metadata syndication format, and an application-level protocol for publishing

© Copyright IBM Corporation 2005. All rights reserved.

and editing Web resources belonging to periodically updated websites.
All Atom feeds must be well-formed XML documents, and are identified with the application/atom+xml media type.

Since well-formedness is of paramount importance in Atom, XSLT is a processing technology that comes quickly to mind. XSLT is a domain-specific language for transforming XML from one format to another, or to text or HTML for presentation. XSLT transforms are an effective cross-platform and cross-language means of processing Atom. This tutorial provides the building blocks for such processing.

Objectives

This tutorial shows you how to navigate the basic structure of Atom 1.0 documents using XPath expressions, how to use these expressions to drive XSLT transformations of Atom source files, and how to deal with the complications of text and markup embedded in Atom files. You will also learn how to use XSLT templates to generate valid Atom files, and how to check the validity of the results.

Prerequisites

This tutorial is written for developers who are familiar with XML, XPath, and XSLT. You should have some familiarity with XHTML and Atom, although the latter is a simple format that you can probably get a good grasp of by looking at the examples early on in this tutorial. See Resources for articles and tutorials that cover these topics. Some of the examples in this tutorial are taken from the background article on Atom 1.0 found in Resources.

System requirements

To run the examples in this tutorial, you need an XSLT processor, preferably one that supports EXSLT (see Resources). I use 4Suite XML 1.0b2. You should also have a recently released Web browser version. When displaying browser output examples, I show screenshots of Firefox 1.0.7 on Fedora Core 4 Linux. Firefox is a popular Web browser available on Windows, Mac OS X, Linux, and other platforms. (Firefox 1.5 is now out, and I strongly encourage you to use that version.)

Section 2. Reading fields from Atom

© Copyright IBM Corporation 2005. All rights reserved.

Atom 1.0 is now Internet Engineering Task Force (IETF) RFC 4287. It's very well defined, so you can be very confident in the structure of Atom documents, and your ability to extract information from them.

Top-level structure and namespace

As with all Atom documents it is a good idea to include the XML declaration, and to specify the encoding. This declaration is transparent to XSLT, and you will be dealing with constructs based on Unicode and the XPath data model. All Atom 1.0 elements are in the namespace `http://www.w3.org/2005/Atom`. Throughout this tutorial, in situations where a prefix is required for this namespace (for example, in XPath expressions) I'll use `a` as the prefix. In many of the code examples, the Atom 1.0 namespace will actually be set up as the default.

Listing 1 (atom-basic.xml) is a complete Atom 1.0 document example.

Listing 1. Atom 1.0 complete example (atom-basic.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
      xml:lang="en"
      xml:base="http://www.example.org">
  <id>http://www.example.org/myfeed</id>
  <title>My Simple Feed</title>
  <updated>2005-07-15T12:00:00Z</updated>
  <link href="/blog" />
  <link rel="self" href="/myfeed" />
  <author><name>Uche Ogbuji</name></author>
  <entry>
    <id>http://www.example.org/entries/1</id>
    <title>A simple blog entry</title>
    <link href="/blog/2005/07/1" />
    <updated>2005-07-14T12:00:00Z</updated>
    <summary>This is a simple blog entry</summary>
  </entry>
  <entry>
    <id>http://www.example.org/entries/2</id>
    <title />
    <link href="/blog/2005/07/2" />
    <updated>2005-07-15T12:00:00Z</updated>
    <summary>This is simple blog entry without a
title</summary>
  </entry>
</feed>
```

Every Atom document contains a top-level `feed` element and zero or more `entry` elements. Every `feed` and `entry` element must include these three elements:

- `id` -- a globally unique identifier
- `title` -- can be empty, although you are strongly encouraged to always provide title content

© Copyright IBM Corporation 2005. All rights reserved.

- `updated` -- a time stamp of the last time the entry or feed was updated

Identifier, title, and updated

The `id` element's content is simple text. You can access the ID of the entire feed using the XPath expression `/a:feed/a:id`. You can get all the entry IDs using `/a:feed/a:entry/a:id`.

The `title` element's content follows Atom's careful conventions for brief, structured text. I'll cover such fields in more detail in the next section. For now just pay attention to the string value of the element, which can be accessed for the entire feed at `/a:feed/a:title`.

The `updated` element's content is a full ISO-8601 date/time point. You can treat this value as simple text for some purposes. See Resources for an article where you can learn more advanced date processing techniques in XSLT. The updated date is accessed for the entire feed at `/a:feed/a:updated`.

Links

An Atom feed will almost always have at least one `link` element, with a `rel` attribute value of `self`. This link provides the authoritative URI for the feed itself. Access the feed's self-link using the XPath `/a:feed/a:link[@rel='self']`. In Listing 1 the feed's self-link has the value `/myfeed`. This is a relative URL, and needs to be turned into an absolute URL for proper processing. Here's where the `xml:base` attribute comes in. Processing software should resolve the relative value against the base (`http://www.example.org`) to get the absolute URL `http://www.example.org/myfeed`. You cannot do this using XPath 1.0 (although XPath 2.0 does include a function `resolve-uri` for this purpose). In some cases, this is OK because you can use the URI base processing in the destination system of your processing. If you are generating HTML, for example, you can keep the URIs relative and just make sure the base URI is expressed in the HTML, as demonstrated in a later section.

Access other links based on the relevant link attributes. Entries usually have a main link (to the Web representation of the entry), and you can usually access this using `a:link[not(@rel)]`.

Note: *URIs in Atom are actually specified as Internationalized Resource Identifiers (IRIs) -- IRIs are defined in RFC 3987 (see Resources). However, in this tutorial I shall stick to the term URI, which is probably more familiar to most readers.*

Authors and contributors

© Copyright IBM Corporation 2005. All rights reserved.

Atom enforces attribution. To be specific, at least one `author` element is required for the feed unless *all* the entries have at least one such element. This may seem a harsh restriction, but it's actually good practice considering how much slicing, dicing, and information re-purposing goes on in the world of episodic Web content. Atom helps ensure that the chain of authorship is maintained. A feed or entry can also have one or more `contributor` elements. Authors and contributors are expressed in Atom as structured elements with a mandatory `name` child (plain text), an optional `uri` child (a URI), and an optional `email` child (plain text e-mail address). Access the name of the feed's overall author, if there is one, using the XPath `a:feed/a:author/a:name`.

Categories

One hot topic you might have heard of is *tagging*. This is the practice of associating metadata in the form of *tags* with episodic Web site entries. Tags are simple words that express a category or relevant subject matter for an item. People can then use tags to form streams of related information, so that you could, for example, combine Weblog entries, scheduled events, photos, audio clips, and more from a public convention. Atom supports such tags in a somewhat enriched form (thank goodness). A feed or entry can have a `category` element that contains a mandatory `term` element and other optional metadata.

I will cover some of the other Atom elements in later sections; I don't cover elements that are less commonly used, and that can generally be processed using techniques presented in this tutorial.

Pulling it together in XSLT

You now know enough about accessing Atom feed elements to make sense of your first XSLT example. Listing 2 (tickertape.xslt.xml) is a transform that summarizes an Atom feed as a simple listing in plain text. You can think of it as a human-readable ticker tape for the feed.

Listing 2. Ticker tape, a simple XSLT file that summarizes key fields in an Atom feed (tickertape.xslt.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:transform  version="1.0"
  xmlns:a="http://www.w3.org/2005/Atom"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>

  <xsl:strip-space elements="*"/>
  <xsl:output method="text"/>

  <xsl:template match="*"/>
```


© Copyright IBM Corporation 2005. All rights reserved.

```
  <xsl:template match="a:feed">
    <xsl:text>Atom Feed: </xsl:text><xsl:value-of select="a:id"/>
    <xsl:text>&#10;</xsl:text>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="a:entry">
    <xsl:text>  ----------------------------------------&#10;</xsl:text>
    <xsl:text>  Feed entry: </xsl:text><xsl:value-of select="a:id"/>
    <xsl:text>&#10;</xsl:text>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="a:title|a:updated">
    <xsl:if test="parent::a:entry">
      <xsl:value-of select="'  '"/>
    </xsl:if>
    <xsl:value-of select="local-name()"/>:<xsl:apply-templates/>
    <xsl:text>&#10;</xsl:text>
  </xsl:template>

</xsl:transform>
```

The first thing you might notice is that I mostly used the *push* style of XSLT, with heavy use of templates. As such, you won't find many of the XPath expressions I've presented so far. Those expressions are still useful in understanding the node structure that is addressed with the XSLT. Atom's re-use of metadata element names for the feed and entries allows me to combine treatment of these into the final template in the listing. Notice my use of `xsl:strip-space`, `xsl:text`, and `
` (the character entity for new line) to control the output text formatting.

Running XSLT examples from a Web browser

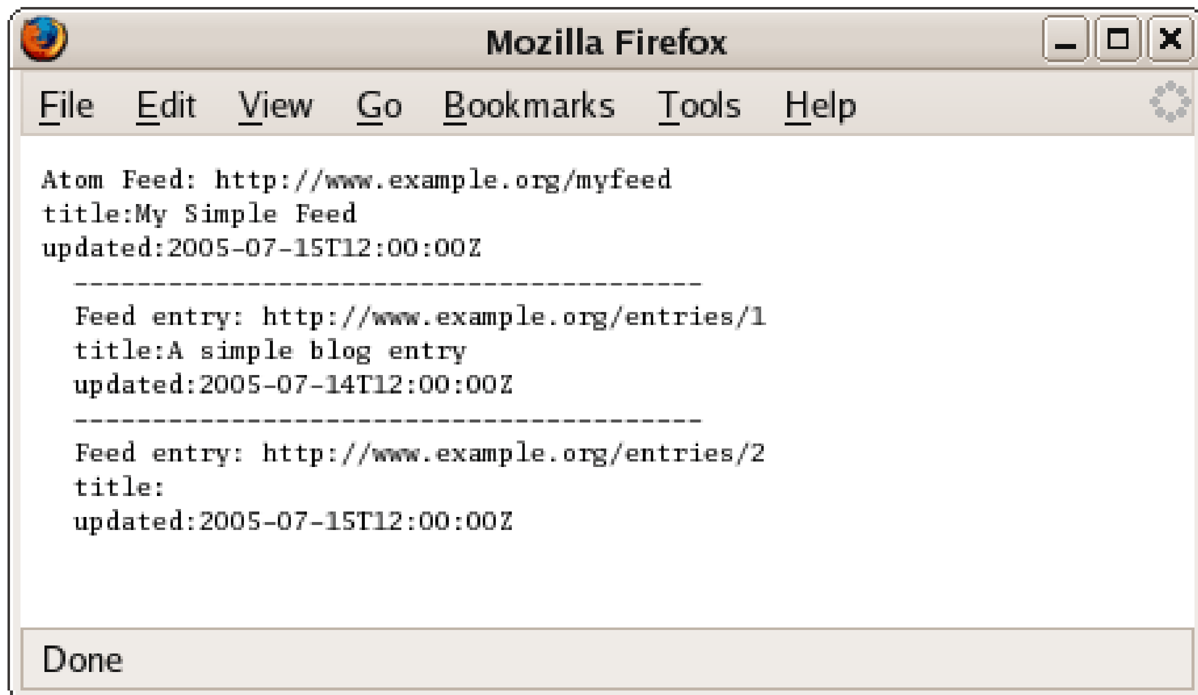
To apply the transform and view the results in a browser, add an XSLT stylesheet processing instruction (PI). atom-basic-1.xml (see Download) is the same as Listing 1 (atom-basic.xml) except that I added such a PI. The first three lines of this version are as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xml" href="tickertape.xslt.xml"?>
<feed xmlns="http://www.w3.org/2005/Atom"
```

I use the extension xslt.xml for Listing 2 because Mozilla-based browsers sometimes have trouble guessing an XML media type for files retrieved from the local disk using the .xsl or .xslt extensions. This results in an error when trying to load the stylesheet. I have found that the most reliable way to avoid such problems is to be sure that the XSLT file ends in .xml. Loading atom-basic-1.xml in a browser, I get the display in Figure 1.

Figure 1. Browser output of Listing 2 applied to Listing 1

© Copyright IBM Corporation 2005. All rights reserved.

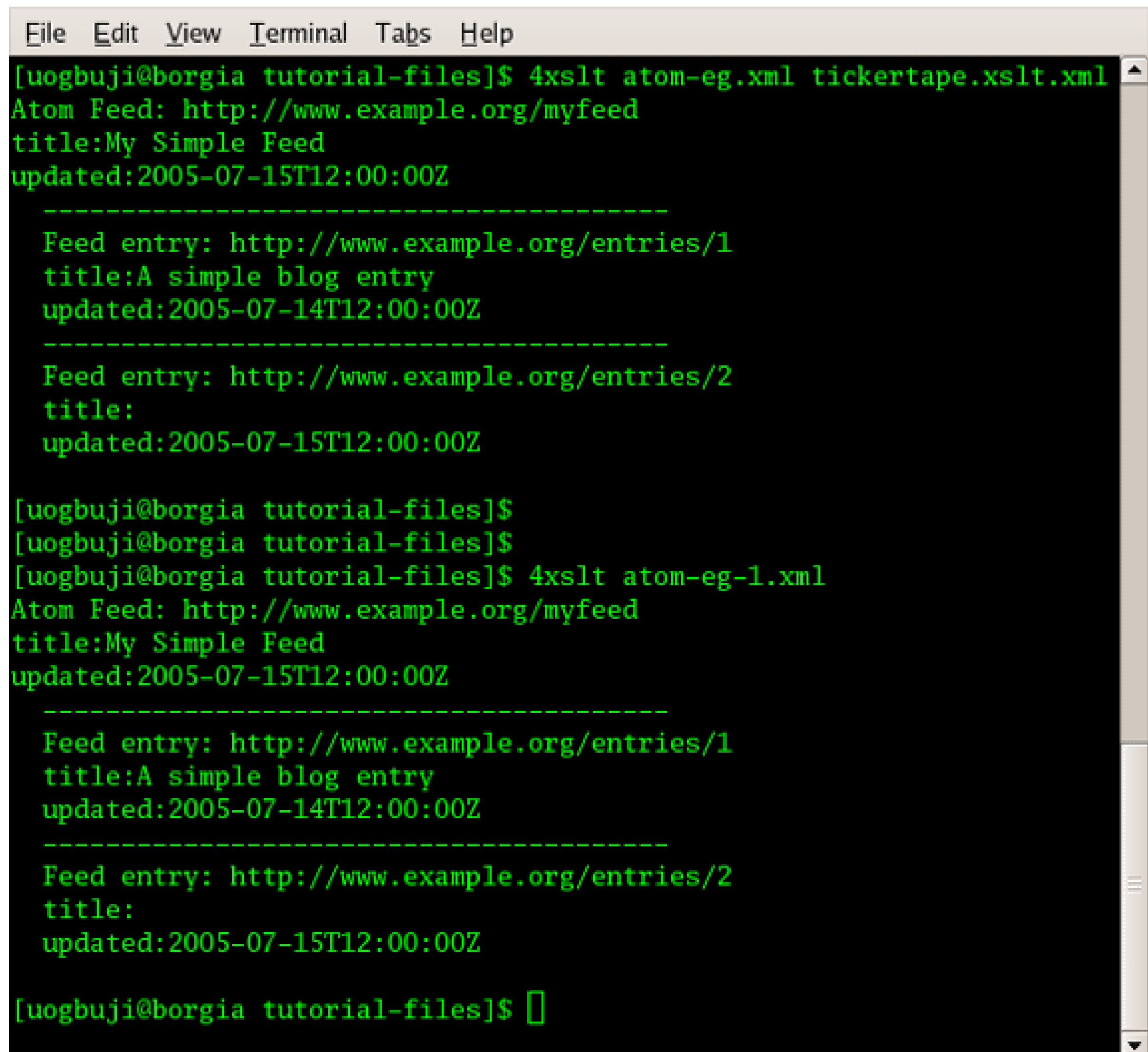


Running XSLT examples from the command line

You can also run this transform using command-line tools. With 4Suite's 4xslt command line tool, for example, you can either specify the plain source file (atom-basic.xml) and the XSLT file (tickertape.xslt.xml), or you can just use the source file with the stylesheet PI (atom-basic-1.xml). Figure 2 shows the console output from both approaches.

Figure 2. Command line session applying Listing 2 to Listing 1

© Copyright IBM Corporation 2005. All rights reserved.



When following along with the examples, you need to decide whether to use a command-line processor or your Web browser. You can also use other tools such as an XSLT IDE, if that is more convenient for you.

Section 3. Processing Atom content fields

At the heart of Atom are the bits of content that give a preview of the Web sites driving the feeds. Because of the chaotic legacy of HTML, it is actually a pretty complex matter to deal with all the ways people might want to represent these bits of content. One of Atom's most important improvements over other Web feed formats

© Copyright IBM Corporation 2005. All rights reserved.

(such as RSS 2.0) is its very clear and careful rules regarding how content can be represented.

A closer look at title

In the previous section, I suggested the temporary simplification that titles are only plain text. This was the case for all titles in Listing 1, but you can have markup in a title as long as you mark it as a richer form of text using the `type` attribute, which defaults to `text`. One option is to set `type="html"` and then use HTML with special characters escaped so that it is valid XML character data, as demonstrated in Listing 3.

Listing 3. Example of a title using HTML

```
<title type="html">
   Making an &lt;strong&gt;emphatic&lt;/strong&gt; statement
</title>
```

Such escaping does make the contained markup a second-class citizen, and can complicate processing accordingly. A much better way to do this is to use embedded XHTML, which is embedded in the Atom without escaping, and is thus easier to process. Listing 4 is similar to Listing 3, but uses an XHTML content construct.

Listing 4. Example of a title using XHTML type

```
<title type="xhtml">
  <div xmlns="http://www.w3.org/1999/xhtml">
    Making an <strong>emphatic</strong> statement
  </div>
</title>
```

The embedded elements must be in the XHTML namespace, which in this case is done by redefining the default namespace. All such content must also be wrapped in a `div` element, which is considered a marker and *not* part of the markup being expressed.

XHTML content example

XHTML's importance makes this content construct ideal for distribution in Atom. Listing 5 is a more complex Atom 1.0 example that uses XHTML text constructs.

Listing 5. Atom 1.0 example using XHTML constructs (atom-with-xhtml.xml)

```
<?xml version="1.0" encoding="utf-8"?>
```


© Copyright IBM Corporation 2005. All rights reserved.

```
<feed xmlns="http://www.w3.org/2005/Atom"
      xml:lang="en"
      xml:base="http://copia.ogbuji.net">
  <id>http://copia.ogbuji.net/atom1.0</id>
  <title>Copia</title>
  <updated>2005-07-15T12:00:00Z</updated>
  <author>
    <name>Uche Ogbuji</name>
    <uri>http://uche.ogbuji.net</uri>
    <email>uche@ogbuji.net</email>
  </author>
  <link href="/blog" />
  <link rel="self" href="/blog/atom1.0" />
  <entry>
    <id>http://copia.ogbuji.net/blog/2005-09-16/xhtml</id>
    <title>XHTML tutorial pubbed</title>
    <link href="http://copia.ogbuji.net/blog/2005-09-16/xhtml"/>
    <category term="xml"/>
    <category term="css"/>
    <category term="xhtml"/>
    <updated>2005-07-15T12:00:00Z</updated>
    <content type="xhtml">
      <div xmlns="http://www.w3.org/1999/xhtml">
        <p>
          <a href="http://www.ibm.com/developerworks/edu/x-dw-x-xhtml-i.html">
            "XHTML, step-by-step"
          </a>
        </p>
        <blockquote>
          <p>Start working with Extensible Hypertext Markup
          Language. In this tutorial, author Uche Ogbuji shows you how to use
          XHTML in practical Web sites.</p>
        </blockquote>
        <p>In this tutorial</p>
        <ul>
          <li>Tutorial introduction</li>
          <li>Anatomy of an XHTML Web page</li>

          <li>Understand the ground rules</li>
          <li>Replace common HTML idioms</li>
          <li>Some practical considerations</li>
          <li>Wrap up</li>
        </ul>
      </div>
    </content>
  </entry>
  <entry>
    <id>http://copia.ogbuji.net/blog/2005-06-21/XSLT___CSS</id>
    <title>XSLT + CSS tutorial pubbed</title>
    <link href="http://copia.ogbuji.net/blog/2005-06-21/XSLT___CSS" />
    <category term="xml"/>
    <category term="css"/>
    <category term="xslt"/>
    <updated>2005-07-14T12:00:00Z</updated>
    <content type="xhtml">
      <div xmlns="http://www.w3.org/1999/xhtml">
        <p>
          <a href="http://www.ibm.com/developerworks/edu/x-dw-x-xmlcss3-i.html">
            "Use Cascading Stylesheets to display XML, Part 3"
          </a>
        </p>
        <blockquote>
          <p>CSS isn't just for HTML anymore! Learn to combine the strengths
             of CSS with those of XSLT and fine-tune your XML presentation
             in a browser.</p>
        </blockquote>
        <ul>
          <li>
          <a href="http://www.ibm.com/developerworks/edu/x-dw-x-xmlcss-i.html">
```


© Copyright IBM Corporation 2005. All rights reserved.

```
          "Use Cascading Stylesheets to display XML, Part 1"</a>
          which introduces the use of CSS to style XML in browsers
          (November 2004)</li>
          <li><a href="http://www.ibm.com/developerworks/edu/x-dw-x-xmlcss2-i.html">
          "Use Cascading Stylesheets to display XML, Part 2"</a>
          which covers advanced topics for the use of CSS to style XML
          in browsers (February 2005)</li>
        </ul>
      </div>
    </content>
  </entry>
</feed>
```

The `content` and `category` elements are also new in this example. The `xhtml` content essentially creates an island of distinct formatting within the Atom substrate. This makes for some challenges in XSLT processing, and provides a strong argument for modularized transforms and push-style processing. I'll demonstrate these practices throughout this tutorial.

Transforming to XHTML

Listing 6 (atom2xhtml.xslt.xml) is a very modular transform from Atom to XHTML 1.1. It includes support for text and XHTML content structures.

Listing 6. XSLT transform providing an XHTML view of an Atom feed (atom2xhtml.xslt.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:a="http://www.w3.org/2005/Atom"
  xmlns:xhtml="http://www.w3.org/1999/xhtml"
  xmlns="http://www.w3.org/1999/xhtml"
  exclude-result-prefixes="a xhtml">

  <xsl:output method="xml" encoding="utf-8"
              doctype-public="-//W3C//DTD XHTML 1.1//EN"
              doctype-system="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"/>

  <xsl:template match="*"/><!-- Ignore unknown elements -->
  <xsl:template match="*" mode="links"/>
  <xsl:template match="*" mode="categories"/>

  <xsl:template match="a:feed">
<html xml:lang="en">
  <head>
    <title><xsl:value-of select="a:title"/></title>
  </head>
  <body>
    <h1><xsl:apply-templates select="a:title" mode="text-construct"/></h1>
    <xsl:apply-templates/>
    <p>Feed ID: <xsl:value-of select="a:id"/></p>
    <p>Feed updated: <xsl:value-of select="a:updated"/></p>
    <xsl:apply-templates/>
  </body>
</html>
  </xsl:template>
```


© Copyright IBM Corporation 2005. All rights reserved.

```
  <xsl:template match="a:summary">
    <div class="summary">
      <xsl:apply-templates select="." mode="text-construct"/>
    </div>
  </xsl:template>

  <xsl:template match="a:content">
    <div class="content">
      <xsl:apply-templates select="." mode="text-construct"/>
    </div>
  </xsl:template>

  <xsl:template match="a:entry">
    <div class="entry">
      <h2><xsl:apply-templates select="a:title" mode="text-construct"/></h2>
      <div class="id">Entry ID: <xsl:value-of select="a:id"/></div>
      <div class="updated">Entry updated: <xsl:value-of select="a:updated"/></div>
      <div class="links">
        <xsl:text/>Links: <xsl:apply-templates select="a:link" mode="links"/>
      </div>
      <div class="categories">
        <xsl:text>Categories: </xsl:text>
        <xsl:apply-templates select="a:category" mode="categories"/>
      </div>
      <xsl:apply-templates/>
    </div>
  </xsl:template>

  <xsl:template match="a:link" mode="links">
    <a href="{@href}">
      <xsl:value-of select="@rel"/>
      <xsl:if test="not(@rel)">[generic link]</xsl:if>
      <xsl:if test="@type">
        <xsl:text> (</xsl:text><xsl:value-of select="@type"/><xsl:text>): </xsl:text>
      </xsl:if>
      <xsl:value-of select="@title"/>
    </a>
    <xsl:if test="position() != last()">
      <xsl:text> | </xsl:text>
    </xsl:if>
  </xsl:template>

  <xsl:template match="a:category" mode="categories">
    <xsl:value-of select="@term"/>
    <xsl:if test="position() != last()">
      <xsl:text> | </xsl:text>
    </xsl:if>
  </xsl:template>

  <xsl:template match="*[@type='text']|*[not(@type)]" mode="text-construct">
    <xsl:value-of select="node()"/>
  </xsl:template>

  <xsl:template match="*[@type='xhtml']" mode="text-construct">
    <xsl:copy-of select="node()"/>
  </xsl:template>

</xsl:stylesheet>
```

First of all, notice that the XHTML namespace is declared twice: once with a prefix, because XPath 1.0 requires the use of a prefix to access elements and attributes within a namespace; and once without a prefix for generating output (so you want to use the default namespace form). By default, XSLT processors copy namespace declarations in the stylesheet to the output. To avoid extraneous prefixes in the output, I use the `exclude-result-prefixes` attribute. The `xsl:output`

© Copyright IBM Corporation 2005. All rights reserved.

instruction is set up to produce XHTML 1.1, including the required document type declaration.

Atom text constructs are handled using a set of templates with the mode `text-construct`. Separate templates handle the case of plain text or XHTML constructs by matching on the attribute. As you can see, the simple difference is that plain text uses `xsl:value-of` to produce a single character data output node, while XHTML uses `xsl:copy-of` to reflect the markup verbatim in the XHTML output. There is no special support in this transform for `type="html"`. Such constructs are very difficult to usefully process, especially with XSLT since the markup is disguised and might not even be well-formed. In this transform, such text constructs are passed verbatim as plain text to the output (a consequence of the XSLT default templates, which I did not override for the `text-construct`) mode). In the case of XHTML output, this means that the tags will be visible in a browser that's used to read the output, rather than turned into presentation discussions. Of course, this might be confusing to anyone who does not know the HTML language.

The lists of links and categories that occur within metadata blocks also use modes for modular rendering. It's very easy to add additional fields in the same way, even for extensions to Atom.

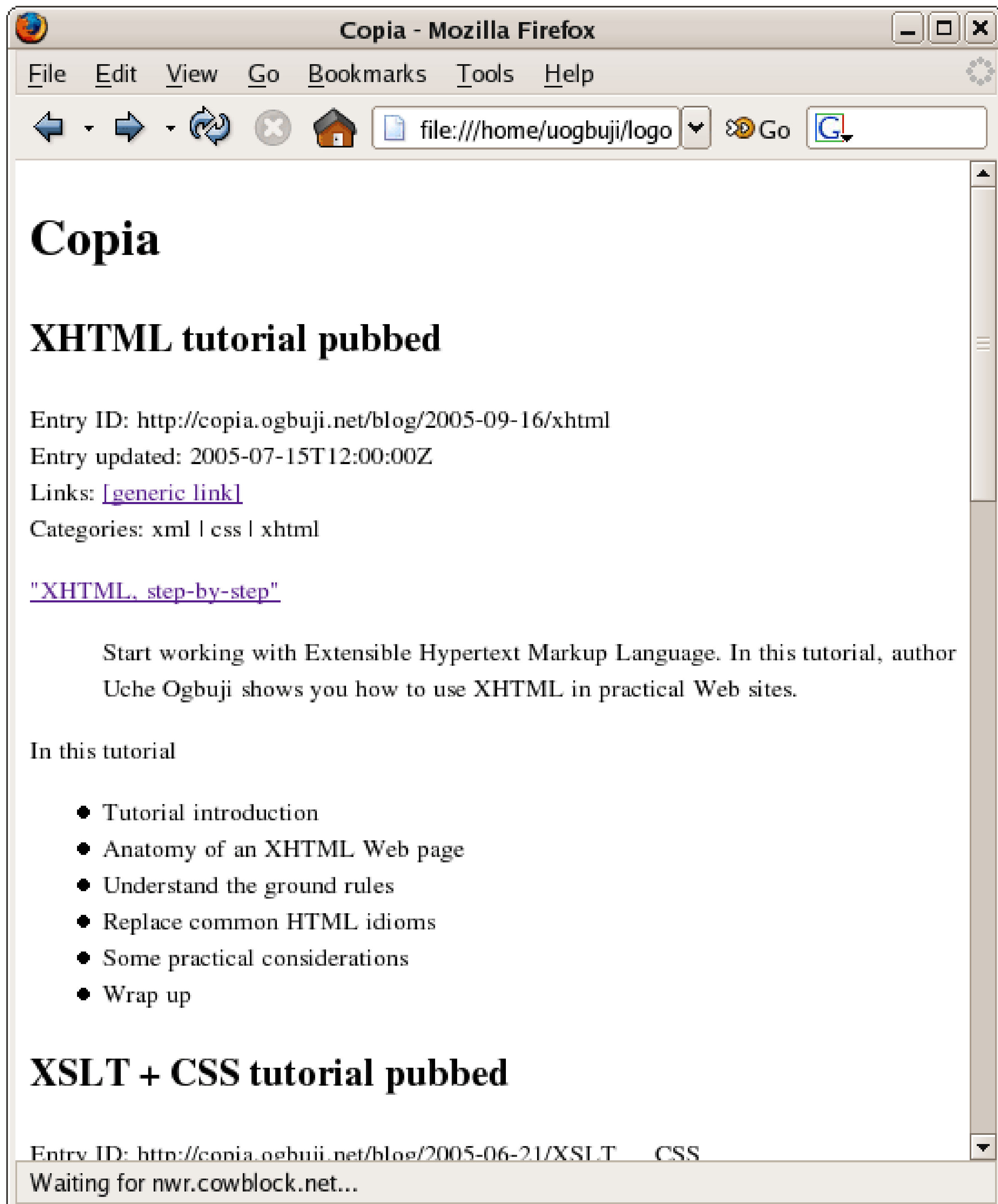
Notice the specification of `xml:lang="en"` for the output. I do this because I have hardcoded English language names for labels and such in the XSLT (Categories, for instance). You will probably want to adapt such labels to your local language and either change the language declaration, or use some automated internationalization technique. Unfortunately, even then you deal with the possibility that the language in a feed you are displaying does not match the declared language for the document as a whole. This is a hard problem, and well beyond the scope of this tutorial, but some might choose to omit a language altogether. On the other hand, in many cases all the information will be in a single language, and in such cases you can and should be sure to specify the language.

Output of the XHTML transform

atom-with-xhtml-1.xml (see Download) is the same as Listing 5 (atom-with-xhtml.xml) except that a PI has been added to render using Listing 6 (atom2xhtml.xslt.xml). The result is the browser view shown in Figure 3.

Figure 3. Browser output of Listing 6 applied to Listing 5

© Copyright IBM Corporation 2005. All rights reserved.



Content constructs

The Atom `content` element can have any of the already mentioned text constructs, but it can also have more. Other text elements are designed to have brief bits of text,

© Copyright IBM Corporation 2005. All rights reserved.

whereas `content` is designed to hold perhaps the entire material in a Weblog entry. Atom affords this element additional flexibility for this reason. The first addition is that the `type` attribute can be set to any Internet media type. If the media type is considered a text media type according to some specific rules, you just put the text into the element body. If the media type is considered an XML media type, you can have XML content of any vocabulary directly in the element body. For any other media type, the content must be Base64 encoded (this might be the case if the content is a picture or audio file). I shall only cover text and XML media types in this tutorial.

`content` elements can also use a `src` attribute to point to content found externally. This attribute contains the URI for the external content, and in this case the `content` element must be empty. This construct is similar to the `object` element in HTML and XHTML.

Listing 7 (atom2xhtml-1.xslt.xml) is based on Listing 6 (atom2xhtml.xslt.xml), but adds support for text and XML content media types.

Listing 7. XSLT in Listing 6 updated to support text and XML media type content (atom2xhtml-1.xslt.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:a="http://www.w3.org/2005/Atom"
  xmlns:xhtml="http://www.w3.org/1999/xhtml"
  xmlns="http://www.w3.org/1999/xhtml"
  exclude-result-prefixes="a xhtml">

  <xsl:output method="xml" encoding="utf-8"
              doctype-public="-//W3C//DTD XHTML 1.1//EN"
              doctype-system="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"/>
  <xsl:template match="*"/><!-- Ignore unknown elements -->
  <xsl:template match="*" mode="links"/>
  <xsl:template match="*" mode="categories"/>

  <xsl:template match="a:feed">
<html xml:lang="en">
  <head>
    <title><xsl:value-of select="a:title"/></title>
  </head>
  <body>
    <h1><xsl:apply-templates select="a:title" mode="text-construct"/></h1>
    <xsl:apply-templates/>
    <p>Feed ID: <xsl:value-of select="a:id"/></p>
    <p>Feed updated: <xsl:value-of select="a:updated"/></p>
    <xsl:apply-templates/>
  </body>
</html>
  </xsl:template>

  <xsl:template match="a:summary">
    <div class="summary">
      <xsl:apply-templates select="." mode="text-construct"/>
    </div>
  </xsl:template>

  <xsl:template match="a:content">
```


© Copyright IBM Corporation 2005. All rights reserved.

```
    <div class="content">
      <xsl:apply-templates select="." mode="text-construct"/>
    </div>
  </xsl:template>

  <xsl:template match="a:entry">
    <div class="entry">
      <h2><xsl:apply-templates select="a:title" mode="text-construct"/></h2>
      <div class="id">Entry ID: <xsl:value-of select="a:id"/></div>
      <div class="updated">Entry updated: <xsl:value-of select="a:updated"/></div>
      <div class="links">
        <xsl:text/>Links: <xsl:apply-templates select="a:link" mode="links"/>
      </div>
      <div class="categories">
        <xsl:text>Categories: </xsl:text>
        <xsl:apply-templates select="a:category" mode="categories"/>
      </div>
      <xsl:apply-templates/>
    </div>
  </xsl:template>

  <xsl:template match="a:link" mode="links">
    <a href="{@href}">
      <xsl:value-of select="@rel"/>
      <xsl:if test="not(@rel)">[generic link]</xsl:if>
      <xsl:if test="@type">
        <xsl:text> (</xsl:text><xsl:value-of select="@type"/><xsl:text>): </xsl:text>
      </xsl:if>
      <xsl:value-of select="@title"/>
    </a>
    <xsl:if test="position() != last()">
      <xsl:text> | </xsl:text>
    </xsl:if>
  </xsl:template>

  <xsl:template match="a:category" mode="categories">
    <xsl:value-of select="@term"/>
    <xsl:if test="position() != last()">
      <xsl:text> | </xsl:text>
    </xsl:if>
  </xsl:template>

  <xsl:template match="*[@type='text']|*[not(@type)]" mode="text-construct">
    <xsl:value-of select="node()"/>
  </xsl:template>

  <xsl:template match="*[@type='xhtml']" mode="text-construct">
    <xsl:copy-of select="node()"/>
  </xsl:template>

  <!-- Handle content elements with external data -->
  <xsl:template match="a:content[@src]" mode="text-construct">
    <!-- Use the XHTML object element -->
    <object data="{@src}" type="{@type}">Unable to render object</object>
  </xsl:template>

  <!-- Handle text media types for content elements -->
  <xsl:template
    match="a:content[not(@src)][starts-with(@type, 'text/')]"
    mode="text-construct">
    <xsl:value-of select="node()"/>
  </xsl:template>

  <!-- Handle XML media types for content elements -->
  <xsl:template
    match="a:content[not(@src)][substring-after(@type, '/') = 'xml'
           or starts-with(substring-after(@type, '/'), '+') = 'xml']"
    mode="text-construct">
    <!-- Just copy the XML over.  If mixed mode XHTML is not supported
```


© Copyright IBM Corporation 2005. All rights reserved.

```
        this might cause problems -->
    <xsl:copy-of select="node()"/>
  </xsl:template>

</xsl:stylesheet>
```

Transforming to HTML 4.01

Not all Web user agents handle XHTML 1.1 well, so you might want to generate HTML output from Atom 1.0. Listing 8 (atom2html4.xslt.xml) does the trick, and includes support for text and XHTML text constructs, but not for content structures expressed by media type. As you might expect, it's very similar to Listing 6 (atom2xhtml.xslt.xml). Adding support for content structures expressed by media type would be a matter of copying code from Listing 7 and changing it according to how you want to deal with difficult cases, such as the incorporation of literal XML in content.

Listing 8. XSLT transform providing an HTML 4 view of an Atom feed (atom2html4.xslt.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:a="http://www.w3.org/2005/Atom"
  xmlns:xhtml="http://www.w3.org/1999/xhtml"
  exclude-result-prefixes="a xhtml">

  <xsl:output method="html" version="4.01" encoding="iso-8859-1"
              doctype-public="-//W3C//DTD HTML 4.01//EN"
              doctype-system="http://www.w3.org/TR/html4/strict.dtd"/>

  <xsl:template match="*"/><!-- Ignore unknown elements -->
  <xsl:template match="*" mode="links"/>
  <xsl:template match="*" mode="categories"/>

  <xsl:template match="a:feed">
<html>
  <head>
    <title><xsl:value-of select="a:title"/></title>
  </head>
  <body>
    <h1><xsl:apply-templates select="a:title" mode="text-construct"/></h1>
    <xsl:apply-templates/>
    <p>Feed ID: <xsl:value-of select="a:id"/></p>
    <p>Feed updated: <xsl:value-of select="a:updated"/></p>
    <xsl:apply-templates/>
  </body>
</html>
  </xsl:template>

  <xsl:template match="a:summary">
    <div class="summary">
      <xsl:apply-templates select="." mode="text-construct"/>
    </div>
  </xsl:template>

  <xsl:template match="a:content">
    <div class="content">
```


© Copyright IBM Corporation 2005. All rights reserved.

```
      <xsl:apply-templates select="." mode="text-construct"/>
    </div>
  </xsl:template>

  <xsl:template match="a:entry">
    <div class="entry">
      <h2><xsl:apply-templates select="a:title" mode="text-construct"/></h2>
      <div class="id">Entry ID: <xsl:value-of select="a:id"/></div>
      <div class="updated">Entry updated: <xsl:value-of select="a:updated"/></div>
      <div class="links">
        <xsl:text/>Links: <xsl:apply-templates select="a:link" mode="links"/>
      </div>
      <div class="categories">
        <xsl:text>Categories: </xsl:text>
        <xsl:apply-templates select="a:category" mode="categories"/>
      </div>
      <xsl:apply-templates/>
    </div>
  </xsl:template>

  <xsl:template match="a:link" mode="links">
    <a href="{@href}">
      <xsl:value-of select="@rel"/>
      <xsl:if test="not(@rel)">[generic link]</xsl:if>
      <xsl:if test="@type">
        <xsl:text> (</xsl:text><xsl:value-of select="@type"/><xsl:text>): </xsl:text>
      </xsl:if>
      <xsl:value-of select="@title"/>
    </a>
    <xsl:if test="position() != last()">
      <xsl:text> | </xsl:text>
    </xsl:if>
  </xsl:template>

  <xsl:template match="a:category" mode="categories">
    <xsl:value-of select="@term"/>
    <xsl:if test="position() != last()">
      <xsl:text> | </xsl:text>
    </xsl:if>
  </xsl:template>

  <xsl:template match="*[@type='text']|*[not(@type)]" mode="text-construct">
    <xsl:value-of select="node()"/>
  </xsl:template>

  <xsl:template match="*[@type='xhtml']" mode="text-construct">
    <xsl:apply-templates select="node()" mode="xhtml2html"/>
  </xsl:template>

  <xsl:template match="*" mode="xhtml2html">
    <xsl:element name="{local-name()}">
      <xsl:apply-templates select="@*|node()" mode="xhtml2html"/>
    </xsl:element>
  </xsl:template>

  <!-- omits comments and PIs -->
  <xsl:template match="@*|text()" mode="xhtml2html">
    <xsl:copy/>
  </xsl:template>

</xsl:stylesheet>
```

The first difference from Listing 6 is in the namespace declarations. There is no default namespace declaration because the output elements are now regular HTML. The `xsl:output` has also changed to use the appropriate XSLT idioms for generating HTML 4. Finally, near the end of the transform you'll now see three

© Copyright IBM Corporation 2005. All rights reserved.

templates for handling XHTML text constructs where there used to be one. This process is now more complex because of the need to omit namespaces and avoid other XMLisms. The final two templates, in the `xhtml2html` mode, are similar to the identity transform, but they copy only elements, attributes, and text to the output, verbatim besides stripping namespaces. Technically, you might also want to change XHTML idioms for language and ID declarations to HTML forms, but these will probably be rare in XHTML content in feeds, anyway.

Section 4. Generating Atom

In the previous section, I showed you XSLT techniques for reading Atom. In this one, you'll see techniques that are mostly useful for creating Atom feeds.

Pulling Atom from XHTML

Atom feeds are usually just a collection of content metadata, and XHTML is one of the leading XML formats for content. As such, you may find yourself wanting to generate Atom directly from XHTML. By schema or convention, XHTML covers optional structures that support almost everything you'd want in an Atom feed. Listing 9 (xhtml2atom.xslt) is a transform that takes a list of URLs for valid XHTML documents, parses them, and generates an Atom feed with an entry for each URL.

Listing 9. Transform to generate an Atom feed from a list of XHTML files (xhtml2atom.xslt)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:transform  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xhtml="http://www.w3.org/1999/xhtml"
  xmlns:a="http://www.w3.org/2005/Atom"
  xmlns="http://www.w3.org/2005/Atom"
  exclude-result-prefixes="a xhtml">

  <xsl:output method="xml" indent="yes"/>
  <xsl:param name="feed-id"
select="'http://www.example.org/dw-feed'"/>
  <xsl:param name="feed-title" select="'Sample Atom feed'"/>
  <xsl:param name="feed-updated"
select="'2005-11-07T12:00:00Z'"/>
  <xsl:param name="main-link"
      select="'http://www.ibm.com/developerworks/'"/>
  <xsl:param name="entries-author" select="'James Snell'"/>

  <xsl:template match="/">
<feed xml:lang="en">
  <id><xsl:value-of select="$feed-id"/></id>
  <title><xsl:value-of select="$feed-title"/></title>
```


© Copyright IBM Corporation 2005. All rights reserved.

```
    <updated><xsl:value-of select="$feed-updated"/></updated>
    <author><name><xsl:value-of
        select="$entries-author"/></name></author>
    <link href="{$main-link}"/>
    <xsl:apply-templates/>
  </feed>
    </xsl:template>

    <xsl:template match="link">
      <xsl:apply-templates select="document(@href)"
  mode="parse-xhtml">
        <xsl:with-param name="entry-href" select="@href"/>
      </xsl:apply-templates>
    </xsl:template>

    <xsl:template match="/" mode="parse-xhtml">
      <xsl:param name="entry-href"/>
      <entry>
        <xsl:apply-templates select="/*/xhtml:head"
  mode="parse-xhtml">
          <xsl:with-param name="entry-href"
  select="$entry-href"/>
        </xsl:apply-templates>
      </entry>
    </xsl:template>

    <xsl:template match="xhtml:head" mode="parse-xhtml">
      <xsl:param name="entry-href"/>
      <id><xsl:value-of select="$entry-href"/></id>
      <title><xsl:value-of select="xhtml:title"/></title>
      <updated><xsl:value-of select="$feed-updated"/></updated>
      <link href="{$entry-href}" type="application/atom+xml"/>
      <xsl:apply-templates
         select="(//xhtml:body//xhtml:p)[1]" mode="parse-xhtml"/>
    </xsl:template>

    <xsl:template match="xhtml:p" mode="parse-xhtml">
      <summary type="xhtml">
        <div xmlns="http://www.w3.org/1999/xhtml">
          <xsl:copy-of select="node()"/>
        </div>
      </summary>
    </xsl:template>

  </xsl:transform>
```

First I define five key parameters. These carry the important fields for the overall feed. They are set up with default values based on example usage of this transform. In the next section you'll find a demonstration of how you might override these values. You would probably update these values in the transform itself, or by overriding the parameters. The source document for this transform is a `links` element with zero or more `link` elements. These elements are matched by the template with `match="link"`. This template loads the document given in the `href` attribute, and then goes into `parse-xhtml` mode in order to write an entry for that URL, passing along the URL that was loaded for later use in the entry's link to "self".

In `parse-xhtml` mode I direct the processing a bit more directly, even though I'm using push XSLT techniques. This is because I am just grabbing a few well-mapped bits of information from each document. Notice that each entry has a link that uses the `entry-href` passed in. Since I know each link is an XHTML document, I can

© Copyright IBM Corporation 2005. All rights reserved.

set the link's `type` attribute accordingly. I use the XPath `(//xhtml:body//xhtml:p)[1]` to grab the first paragraph in the body of the document for use in a `summary` element. The template with `match="xhtml:p"` recursively copies this element's contents within an XHTML text construct, switching the default namespace to XHTML within the body of the text construct.

The Atom generator in action

Listing 10 (xhtml-urls.xml) is a simple list of XHTML URIs -- in this case three recent IBM developerWorks articles on Atom 1.0, which are plumbed for details to put in an Atom feed.

Listing 10. List of XHTML files for generating an Atom feed (xhtml-urls.xml)

```
<links>
  <link href="http://www-128.ibm.com/developerworks/xml/library/x-atom10.html"/>
  <link href="http://www.ibm.com/developerworks/xml/library/x-extatom1/"/>
  <link href="http://www-128.ibm.com/developerworks/xml/library/x-extatom2.html"/>
</links>
```

And Figure 4 shows the transform in action.

Figure 4. Command-line session applying Listing 9 to Listing 10

© Copyright IBM Corporation 2005. All rights reserved.

```
File  Edit  View  Terminal  Tabs  Help
[uogbuji@borgia tutorial-files]$ 4xslt xhtml-urls.xml xhtml2atom.xslt
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xml:lang="en">
  <id>http://www.example.org/dw-feed</id>
  <title>Sample Atom feed</title>
  <updated>2005-11-07T12:00:00Z</updated>
  <author>
    <name>James Snell</name>
  </author>
  <link href="http://www.ibm.com/developerworks/"/>
  <entry>
    <id>http://www-128.ibm.com/developerworks/xml/library/x-atom10.html</id>
    <title>An overview of the Atom 1.0 Syndication Format</title>
    <updated>2005-11-07T12:00:00Z</updated>
    <link href="http://www-128.ibm.com/developerworks/xml/library/x-atom10.html"
 type="application/atom+xml"/>
    <summary type="xhtml">
      <div xmlns="http://www.w3.org/1999/xhtml">
        <em>How this popular Web content syndication format stacks up</em>
      </div>
    </summary>
  </entry>
  <entry>
    <id>http://www.ibm.com/developerworks/xml/library/x-extatom1/</id>
    <title>Atom 1.0 extensions, Part 1: Feed history, ordering entries, and expi
ration timestamps</title>
    <updated>2005-11-07T12:00:00Z</updated>
    <link href="http://www.ibm.com/developerworks/xml/library/x-extatom1/" type=
"application/atom+xml"/>
    <summary type="xhtml">
      <div xmlns="http://www.w3.org/1999/xhtml">
        <em>An overview of proposed extensions to the Atom 1.0 Syndication Forma
t</em>
      </div>
    </summary>
  </entry>
  <entry>
    <id>http://www-128.ibm.com/developerworks/xml/library/x-extatom2.html</id>
    <title>Atom 1.0 Extensions, Part 2: Copyright licenses, automated processing
 of links, and syndicating threads</title>
    <updated>2005-11-07T12:00:00Z</updated>
    <link href="http://www-128.ibm.com/developerworks/xml/library/x-extatom2.htm
l" type="application/atom+xml"/>
    <summary type="xhtml">
      <div xmlns="http://www.w3.org/1999/xhtml">
        <em>An overview of proposed extensions to the Atom 1.0 Syndication Forma
t</em>
      </div>
    </summary>
  </entry>
</feed>
[uogbuji@borgia tutorial-files]$
```


© Copyright IBM Corporation 2005. All rights reserved.

You can always override the parameters to adjust key fields. So for example, using 4Suite you could change the feed title with an invocation such as:

```
4xslt -D feed-title="IBM dW on Atom" xhtml-urls.xml xhtml2atom.xslt
```

Setting the date

You might have noticed that I hard-coded a date in the above transform (Listing 9). What you probably want is the current date. You can't set this in stock XSLT, but you can using EXSLT, the community standard for XSLT extensions. The following snippet (Listing 11) is an alternative version of the top of Listing 9 that shows how you would set the current date.

Listing 11. First lines of a variation of Listing 9 that sets current date

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:transform  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xhtml="http://www.w3.org/1999/xhtml"
  xmlns:a="http://www.w3.org/2005/Atom"
  xmlns="http://www.w3.org/2005/Atom"
  xmlns:date="http://exslt.org/dates-and-times"
  exclude-result-prefixes="a xhtml">

  <xsl:output method="xml" indent="yes"/>
  <xsl:param name="feed-id" select="'http://www.example.org/dw-feed'"/>
  <xsl:param name="feed-title" select="'Sample Atom feed'"/>
  <xsl:param name="feed-updated" select="$now"/>
  <xsl:variable name="now" select="date:date-time()"/>
```

The last two lines are key. Never mind that the `now` variable is defined later in the file than it is used. XSLT allows this, and the variables will be resolved in appropriate order. See also the added namespace declaration for the `date` prefix. The file xhtml2atom-1.xslt (see Download) is Listing 9 updated with this date code. It's designed to work with an XSLT processor that supports the EXSLT date module; many stand-alone processors such as 4Suite's do this, but unfortunately Mozilla (and thus Firefox) does not.

Section 5. Practical notes

In this section I gather some practical observations that can be especially useful when processing Atom with XSLT.

© Copyright IBM Corporation 2005. All rights reserved.

Handling base URIs

Like many XML formats, Atom allows the user to set a base URI against which relative URIs within the document are resolved. You do this using the `xml:base` attribute. In Listing 1, I used such an attribute on the `feed` element. If you used a transform on it, such as that in Listing 6, you would end up with XHTML containing links such as `/blog`. The Atom has a base URI of `http://www.example.org` in effect, so the intended full URI is `http://www.example.org/blog`. The problem is that if you were to host the resulting XHTML at the URL `http://test.example.org` that relative link would be expanded to `http://test.example.org/blog`, which is no longer the same link.

It would be nice if all you had to do to correct this situation was to copy any `xml:base` attributes from the Atom to the XHTML, but this attribute is unfortunately not valid in XHTML 1.0 or 1.1. It is valid in current drafts of XHTML 2.0, but that specification is still a work in progress. Mozilla browsers do look for `xml:base` in XHTML 1.x, but this is non-standard behavior and you should avoid it. The solution is to create an XHTML `base` element. This is much easier if the Atom source has one `xml:base` attribute at the top level (meaning, on the `feed` element), as in Listing 1. The following snippet (Listing 12) is the relevant template from Listing 6 updated to handle the base URI.

Listing 12. Template for processing Atom feed element updated with base URI support (excerpt from atom2xhtml.xslt-1.xml)

```
<xsl:template match="a:feed">
<html xml:lang="en">
  <head>
    <title><xsl:value-of select="a:title"/></title>
    <base href="{@xml:base}"/>  <!-- handle Base URI -->
  </head>
  <body>
    <h1><xsl:apply-templates select="a:title"
mode="text-construct"/></h1>
    <xsl:apply-templates/>
    <p>Feed ID: <xsl:value-of select="a:id"/></p>
    <p>Feed updated: <xsl:value-of select="a:updated"/></p>
    <xsl:apply-templates/>
  </body>
</html>
</xsl:template>
```

The file atom2xhtml.xslt-1.xml is Listing 6 updated with this base URI code (see Download).

Validating your work

In this tutorial you've learned a lot about how to process Atom 1.0, whether reading

© Copyright IBM Corporation 2005. All rights reserved.

or writing the format. You'll find this much easier if you can be sure that the Atom documents in question are valid. However, everyone makes mistakes, and there is no substitute for checking for such correctness. I recommend validating incoming and outgoing Atom at least in the testing phases of a project, and making it easy to turn validation back on again in production if problems arise. You can do a degree of testing using the RELAX NG schema that's associated with the specification (see Resources). To do so, you need a RELAX NG processor that reads the compact syntax, or a tool that you can use to convert from the compact to the XML syntax.

Using the W3C's online validation form

RELAX NG cannot check all the constraints placed on Atom documents. Luckily the Atom community has developed validation tools, and the W3C hosts a form-based interface for Atom validation. You can either enter an address for an Atom document hosted online ("Validate by URL"), or paste in the contents of the document ("Validate by Direct Input"). If you use the former method, the validator also checks your Web server setup for the Atom document -- for example verifying that the character encoding reported by the server matches what is reported in the document. You can also send documents for validation by HTTP POST or SOAP. Figure 5 shows a sample browser session using the validator.

Figure 5. Browser session using the W3C feed validator

© Copyright IBM Corporation 2005. All rights reserved.

Feed Validator Results: - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://validator.w3. Go feedvalidator.org

W3C® QUALITY Assurance FEED Validation Service

Home About... News Docs

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <id>http://copia.ogbuji.net/atom1.0</id>
  <title>Copia</title>
  <updated>2005-07-15T12:00:00Z</updated>
  <author>
    <name>Uche Ogbuji</name>
    <uri>http://uche.ogbuji.net</uri>
    <email>uche@ogbuji.net</email>
  </author>
  <link href="/blog" />
  <link rel="self" href="/blog/atom1.0" />
  <entry>
    <id>http://copia.ogbuji.net/blog/2005-09-16/xhtml</id>
    <title>XHTML tutorial pubbed</title>
```

Validate

Warning

This feed is valid, but may cause problems for some users. We recommend fixing these problems.

- line 12, column 42: Self reference doesn't match document location

Done

The W3C's validator is based on code originally hosted at `feedvalidator.org`. The W3C recently launched its own service in late November, and the system still has some kinks. I hope these will be cleared up by the time you read this, so you can reliably use the W3C feed validator.

© Copyright IBM Corporation 2005. All rights reserved.

Section 6. Wrap up

Summary

In this tutorial, you have learned how to process Atom 1.0 using XSLT 1.0. In particular you have learned how to:

- Access the basics of Atom format
- Interpret and access links in Atom
- Interpret and access author and category information in Atom
- Generate a simple ticker tape summary of an Atom feed
- Set up the ticker tape transform in a browser or stand-alone XSLT processor
- Process plain text Atom text fields
- Process XHTML Atom text fields
- Transform Atom feeds to XHTML and HTML 4
- Pull information from XHTML to populate an Atom feed
- Set the current date in an Atom feed
- Preserve base URI information from Atom to XHTML output
- Check Atom documents with the RELAX NG specification and W3C validator

This is a foundation on which you can build additional processing capabilities. In some cases XHTML 1.0 will be sufficient. In other cases you might need EXSLT or even application processing, but I hope you now have some appreciation of the range of what you can do with well-designed XSLT in processing Atom.

© Copyright IBM Corporation 2005. All rights reserved.

Downloads

Description	Name	Size	Download method
Sample application	x-atomxsl-tutorial-files.zip	11KB	HTTP

Information about download methods

© Copyright IBM Corporation 2005. All rights reserved.

Resources

Learn

- Not familiar with XML? Start with the many helpful resources in the developerWorks "New to XML" page, especially the tutorial "Introduction to XML" by Doug Tidwell (August 2002).
- Get properly introduced to Atom by reading "An overview of the Atom 1.0 Syndication Format", by James Snell (developerWorks, August 2005). Also bookmark the main Atom home page, The Atom 1,0 specification, and the RELAX NG for Atom 1.0.
- Understand XHTML and what sets it apart from HTML in the tutorial "XHTML, step-by-step", by Uche Ogbuji (developerWorks, September 2005).
- Learn about XML namespaces in the article "Plan to use XML namespaces, Part 1" by David Marston (developerWorks, April 2004), and learn more about how to use namespaces effectively in the articles "Plan to use XML namespaces, Part 2" by David Marston (developerWorks, April 2004), and "Principles of XML design: Use XML namespaces with care" by Uche Ogbuji (developerWorks, April 2004).
- Learn more about EXSLT, the community standard for useful and widely supported XSLT extension functions and elements. A good place to start is "EXSLT by example" by Uche Ogbuji (developerWorks, February 2003). That article explores, among other things, techniques for time and date processing using the Date and Time module, used in this tutorial.
- If you're unfamiliar with XPath and XSLT, take a look at "Get started with XPath" by Bertrand Portier (developerWorks, May 2004), move on to "Investigating XSLT: The XML transformation language by LindaMay Patterson (developerWorks, August 2001), and finish up your introductory tour with "Create multi-purpose Web content with XSLT" by Nicholas Chase (developerWorks, March 2003).
- You may want to use RELAX NG as a validation step in your Atom processing. Learn RELAX NG by reading "Understanding RELAX NG", by Nicholas Chase (developerWorks, December 2003). Also, in "XML Matters: Kicking back with RELAX NG, Part 1" (developerWorks, February 2003), Columnist David Mertz gives an overview of the schema language, and then provides more depth in Part 2 (March 2003) and Part 3 (May 2003).
- Learn about IRIs, the internationalized type of URI used in Atom. The W3C has an overview page on Internationalized Resource Identifiers. The full specification is RFC 3987: Internationalized Resource Identifiers (IRIs).
- Peruse the developerWorks XML and Web architecture zones for more information on the technologies covered in this tutorial.

© Copyright IBM Corporation 2005. All rights reserved.

Get products and technologies

- Keep up with developerWorks through Atom feeds. You can access pre-defined feeds or define your own custom feeds.
- Check your Atom documents using the W3C Feed Validation Service, which is based on the technology at feedvalidator.org.
- Follow along by running the tutorial examples yourself. I tested the browser examples in this tutorial with Firefox, a popular and free Web browser based on Mozilla and available on Windows, Mac OS X, Linux, and other platforms. I tested stand-alone XSLT processing on the command-line using 4Suite.

Discuss

- Find tips, tricks, and answers about Atom, RSS, or other syndication topics in the Atom and RSS forum.
- Participate in developerWorks blogs and get involved in the developerWorks community.

About the author

Uche Ogbuji

Uche Ogbuji is a consultant and co-founder of Fourthought Inc., a software vendor and consultancy specializing in XML solutions for enterprise knowledge management. Fourthought develops 4Suite, an open source platform for XML, RDF, and knowledge-management applications. Mr. Ogbuji is also a lead developer of the Versa RDF query language. He is a computer engineer and writer born in Nigeria, living and working in Boulder, Colorado, USA. You can find more about Mr. Ogbuji at his Weblog Copia, or contact him at uche.ogbuji@fourthought.com.

© Copyright IBM Corporation 2005. All rights reserved.