

Automate XML file updates, Part 1: XML process introduction and conversion stylesheet creation

A methodology using XSLT, Apache Ant, and Java SE

Skill Level: Intermediate

[Tom Coppedge \(tcoppedg@us.ibm.com\)](mailto:tcoppedg@us.ibm.com)
developerWorks Software Engineer
IBM

17 Aug 2006

This is the first part of a tutorial series that describes a method for automating updates to a library of XML files so that they all conform to an updated XML schema. In Part 1, you learn the steps in the entire process and then create an XSLT stylesheet to update the XML files. In [Part 2](#), you learn how to install, configure, and run Apache Ant and Java SE to iteratively transform each of your XML files based on the updates specified in your XSLT stylesheet.

Section 1. Before you start

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

About this tutorial

If you're responsible for maintaining a library of schema-based XML, it's likely that the schema upon which the files are based has been updated over time. The schema updates might have been necessary because of updated internal standards at your company or because of the need for your business to carry less, different, or more information within the XML files.

Whatever the cause of the schema updates, you'll want to seriously consider

updating all of your XML files whenever the schema is updated so that they all validate against the current schema. By doing so, the task of processing the XML files is made easier than if the XML files conformed to several schemas. Why? If every file conforms to the same schema, you only have to write one version of code to process those XML files (in this tutorial, it'll be XSLT stylesheets) because it can assume a homogeneous XML structure across the entire library of files. Mass conformance to one schema is especially important if your new schema incorporates renamed, added, or removed *required* elements or attributes.

Updating few XML files by hand is no problem. But what if you have hundreds, or thousands, of XML files to update? A programatic solution is required. This tutorial will help you tackle the XML file updates with a methodology that has been proven many times by the developerWorks staff.

Objectives

The objectives for this tutorial series are addressed in the following manner:

Part 1

- Review a checklist for the entire process.
- Create a *conversion stylesheet* -- an XSLT stylesheet with templates to add, update, and delete elements and attributes in existing XML files so that the files conform to a new XML schema.

Part 2 ([Go directly to Part 2 now.](#))

- Install and configure Ant and Java SE.
- Using Ant, iteratively update and validate each of your XML files based on the updates specified in your XSLT stylesheet, and then transform them again to HTML.

Prerequisite knowledge

To get the most from this tutorial, you should be able to create XML schemas (or document type definitions, also known as DTDs) and XSLT stylesheets. See [Resources](#) for developerWorks articles and tutorials that will help you learn these skills.

System requirements

To process the [sample code](#) supplied with this tutorial, install the following software on your computer:

- Microsoft Windows 2000 or later.
- A Web browser.
- A validating XML editor for editing XML schemas, XML instance documents, and XSLT stylesheets.
- The Apache Software Foundation's *Ant* software.
- J2SE™ or J2EE™ (Runtime Environment or Development Kit) 1.2 or higher is required by Ant. Version 1.4 is recommended because it comes with an XSL transformer; therefore, you won't have to download and install Xalan-Java separately, for example. Note: Ant does not support the Microsoft JVM/JDK.

See [Resources](#) for more information.

Section 2. The conversion process: A quick reference

Here's a summary of the entire process. After you read the tutorial, you might want to return to the following table and use it as a quick reference as you plan, schedule and do the work.

Note: You can apply the process described in this tutorial series to XML instance documents based on a document type definition (DTD), as well as those based on a schema. For brevity, I refer only to schema-based documents. The process is applied to the XML instance documents, not the schema or DTD.

Table 1. Process steps

Step	Description
1	Obtain copies of the former schema and the updated schema in its final, approved state.
2	Compare schema files and other update sources. (Part 1)
3	Determine whether an XML file update is required. (Part 1)
4	Create and test XSLT conversion templates. (Part 1)
5	Create and test the XSLT conversion stylesheet. (Part 1)
6	Apache Ant: Introduction and installation instructions. (Part 2)
6	Download and install Java SE. (Part 2)

- | | |
|---|--|
| 7 | Create a build.xml file for Ant. (Part 2) |
| 8 | Run Ant to create new XML and HTML files. (Part 2) |

Section 3. Compare schema files and other update sources

The two drivers of XML instance document updates are:

Schema updates

A schema construct might have been added, changed, or removed. Construct data values that are controlled by the schema (such as enumeration lists) are included in this category as well.

XML data value updates not controlled by the schema

Depending on your application, your XML authors or editors might have established data value guidelines for certain constructs. For whatever reason, those guidelines might not be enforced by the schema and might change over time, requiring the entry of different data values.

Schema updates

Ideally, if you work with other developers, they will have prepared a list of schema changes to help you begin your work. Otherwise, you need to compare the former and updated schema files to determine what has changed.

Fortunately, editing software usually offers a file comparison function. You simply identify the two files to compare, select the characteristics of the comparison (such as whether the comparison engine should differentiate capital from lower case letters, or ignore spaces), and start the function. XML Spy includes such a function, as does Ultraedit (see [Resources](#)). The following figure illustrates a portion of XML Spy's schema file comparison output where the differences between the schema file on the right and left are highlighted in green:

Figure 1. Example schema file comparison output

```

<xsd:element name="employee">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name"
type="xsd:string" minOccurs="1"/>
      <xsd:element name="dept"
type="xsd:string" minOccurs="1" />
      <xsd:element name="
location" type="xsd:string"
minOccurs="1" />
      <xsd:element name="project
" type="xsd:string" maxOccurs="
unbounded" />
      <xsd:element name="phone"
type="xsd:string" minOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>

```

```

<xsd:element name="employee">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="
xsd:string" minOccurs="1"/>
      <xsd:element name="project"
type="xsd:string" maxOccurs="unbounded"
/>
      <xsd:element name="manager"
type="xsd:string" minOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="dept" type="
department" use="required" />
  </xsd:complexType>
</xsd:element>
<!-- Element: dept -->
<xsd:simpleType name="department">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="01"/>
    <xsd:enumeration value="02"/>
    <xsd:enumeration value="03"/>
  </xsd:restriction>
</xsd:simpleType>

```

Of course, if you do a manual comparison, human error will likely cause you to miss some changes. The exposure is greater if multiple developers made changes.

XML data value updates

By design, or by unfortunate necessity, it's possible that your schema does not control certain possible data values for constructs in your XML instance documents. No matter the reason, your XML updates might need to reflect changes to data values in this category.

Here's a real-life example, taken from my own experience at developerWorks. My example illustrates why you might have inherited non-schema-controlled data values. Originally, all of the article and tutorial content on developerWorks was written in HTML. When we made the decision to convert this content to XML (one of the goals being to structure the source more consistently), we discovered a staggering variation in how that content was being structured, and how very similar or identical data values for those constructs were being entered. Each author and editor had a preferred style, and the HTML structure allowed those style variations to pass through unchecked. At this point, we only had about 1500 unique articles and tutorials published on the site. (Today we have over 12,000, in several languages. I'm glad we switched to XML when we did!) So we could complete our HTML-to-XML changeover that year, we had to accept that we could not create enumeration lists for several types of data that we would have if starting from scratch. In summary, we had to define some constructs as string types, to allow for the existing data value variation, and rely on editorial help to establish guidelines for those values from that time forward to limit the variability as much as possible.

My developerWorks example is probably similar to examples in many organizations,

possibly yours. What this means in an XML update situation is this: You must try to find out if guidelines for these non-schema-controlled data values have changed, and account for them in your conversion stylesheet so the data values are updated accordingly.

By noting the updates highlighted in a schema file comparison, and the updates to non-schema-controlled data values, you can create your own list of additional, changed, resequenced, or deleted schema elements, attributes, groups, complexTypes, simpleTypes, and other schema constructs, as well as data values to be updated.

Section 4. Determine whether an XML file update is required

Some schema updates will require you to update your XML instance documents to ensure they conform to the new schema. Other updates require no action and can be ignored.

The following checklist will help you to determine the types of schema updates that will need to be reflected in your XML files. Note: Updates can affect elements, attributes, simpleTypes, complexTypes, and groups; the schema update descriptions in the table apply to any of these constructs.

Table 2: Schema updates requiring XML instance document updates

Schema update description	Action required
Updates to constructs (elements, attributes, simpleTypes, complexTypes, groups)	
A required construct was added	You must add the new required construct to the XML instance documents.
An optional construct was added	No action is required. If you have data values for the optional construct that <i>could</i> be used, you might consider adding the new construct to the XML instance documents if you have time.
A required or optional construct was removed	You must remove the construct from the XML instance documents.
A previously required construct is now optional	No action is required.
A previously optional construct is now required	You must add the new required construct to those XML instance documents that don't already have it.

A construct was renamed or resequenced	You must rename or resequence the construct within the XML instance documents that contain it.
Updates to construct data values (values of elements, attributes, simpleTypes, complexTypes, groups)	
A data value was added to an enumeration list	No action is required.
A data value was deleted from an enumeration list	You must delete that value wherever it occurs in your XML instance documents. If the construct with this data value is required, you must determine an acceptable data value to be used instead of the old one.
A data value has been changed in an enumeration list	You must change that value accordingly wherever it occurs in your XML instance documents.
General note: If an update affects a simpleType or complexType construct, the elements or attributes based on this construct might also be affected. For example, if three elements get their definition from the same complexType construct, and the complexType definition was updated with a new required attribute, then those three elements must be updated with the new attribute wherever they appear in your XML documents.	

If you have multiple XML instance documents that adhere to different versions of the schema (or different schemas), you'll need to address the possible differences between the updated schema and the other versions. Imagine that situation for a minute. It shouldn't take long for you to see the benefit of keeping all the XML documents compliant with the same (current) schema when schema updates are necessary.

By applying this checklist to each schema update you listed in the previous step, you have the basis for creating the XSLT stylesheet templates necessary to update your XML instance documents.

Section 5. Create and test XSLT conversion templates

Once you list the schema updates and the non-schema-controlled data updates, and determine which need to be reflected in your XML instance documents, you can begin writing what I'll call a conversion stylesheet. You'll be converting XML instance documents into new XML instance documents that conform to the updated schema. The process is simple if you take it one step, or XSLT template, at a time.

The template writing process

For each schema update, or each non-schema-controlled data update that requires an XML file update:

1. Write a corresponding XSLT template to reflect that update in the XML instance documents. The template will likely begin by matching (`<xsl:template match=" ">`) a given construct; the rest of the template will be dedicated to outputting XML as intended by the item in the change list. If your development process includes tracking numbers associated with one or more of these changes, you might consider adding a comment to the template with the tracking number to ensure you maintain a history trail.
2. Test the XSLT template. Simply create a one-template stylesheet and transform a representative sample of XML instance documents that will test the functions of the template. If the previous schema definition for a given construct allowed for wide variation in structure and data values, be sure your tests cover all of those cases.
3. Change your XSLT template to reflect any problems testing exposes.

A tale of two templates

The scenario and templates described below illustrate some common conversion stylesheet tasks, such as adding, removing, and changing constructs and data values within an XML file.

Imagine that you maintain a schema that describes employee information. The current employee element within the schema is described by a sequence of the following elements, along with their min/max occurrence indicators:

- Name (min 1, max 1)
- Department (min 1, max 1)
- Location (min 1, max 1)
- Project (min 1, max unbounded)
- Phone (min 1, max 1)

Now imagine that, for various reasons, you or someone else will have to update the schema (and therefore the XML instance documents upon which it's based) as follows:

- Replace the dept element with a new dept attribute on the employee element.
- Replace the old dept values with new ones.
- Delete the location element.
- Add a manager element (based on dept number.)
- Replace the '123' area code with '333'.

All of these changes will require you update the existing XML instance documents so they validate when checked against the new schema.

You can divide the requested changes into two templates: One to handle the area code updates for the phone numbers, and one to handle the rest. In real life, the developer responsible for maintaining the phone information (and the corresponding section of the schema) may be separate from those who maintain the other information, so it would be logical to divide the work this way.

Whatever the justification, here are two templates that implement the required updates. The templates are coded with some assumptions to make them relatively brief. (As any XSL coder will tell you, there are many ways to code a template.) This isn't an XSLT tutorial, so I won't go into any detail about the templates themselves. (See [Resources](#) for very popular XSLT articles and tutorials on developerWorks.) I have, however, added **bold** highlighting to lines containing or introducing the primary functions.

Listing 1. Example conversion template: Updates to the employee element

```
<!-- Change employee element as follows:
  1. Replace dept element with new dept attribute.
  2. Replace old dept numbers with new ones.
  3. Delete location element.
  4. Add manager element.
-->
<xsl:template match="employee">
  <xsl:element name="employee">
    <!-- Add dept attribute; source from dept element. -->
    <xsl:attribute name="dept">
      <xsl:choose>
        <!-- Replace old dept numbers with new values -->
        <xsl:when test="dept='012'">
          <xsl:text>01</xsl:text>
        </xsl:when>
        <xsl:when test="dept='123'">
          <xsl:text>02</xsl:text>
        </xsl:when>
        <xsl:when test="dept='456'">
          <xsl:text>03</xsl:text>
        </xsl:when>
        <xsl:when test="dept='789'">
          <xsl:text>04</xsl:text>
        </xsl:when>
        <xsl:otherwise>
```

```

        <xsl:text>05</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:attribute>
</xsl:for-each select="*">
  <xsl:choose>
    <!-- Delete dept and location elements. -->
    <xsl:when test="name()='dept' or name()='location'"/>
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="name()='phone'">
          <!-- The match="phone" template will be called -->
          <xsl:apply-templates select="."/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:element name="{name()}">
            <xsl:value-of select="."/>
          </xsl:element>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:for-each>
<!-- Add manager element -->
<xsl:element name="manager">
  <xsl:choose>
    <xsl:when test="dept='012'">
      <xsl:text>Ms. Alpha</xsl:text>
    </xsl:when>
    <xsl:when test="dept='123'">
      <xsl:text>Mr. Bravo</xsl:text>
    </xsl:when>
    <xsl:when test="dept='456'">
      <xsl:text>Mr. Charlie</xsl:text>
    </xsl:when>
    <xsl:when test="dept='789'">
      <xsl:text>Ms. Delta</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>Ms. Parker</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:element>
</xsl:element>
</xsl:template>

```

Listing 2. Example conversion template: Updates to the phone element

```

<!-- Change phone element as follows:
      Replace '123' area code with '333'.
-->
<xsl:template match="phone">
  <!-- Change area code 123 to 333, but keep the exchange+number -->
  <xsl:variable name="exchange-number">
    <xsl:value-of select="substring-after(.,'123')"/>
  </xsl:variable>
  <xsl:element name="phone">
    <xsl:value-of select="concat('333', $exchange-number)"/>
  </xsl:element>
</xsl:template>

```

Additional benefits of this process

By addressing each change with a separate template, you gain the ability to divide the work among several XSLT developers. Each developer can code and test the assigned list of updates independently, thereby adding some flexibility to the schedule and work to be done. If you are able to divide up the work, try to do it so your developers won't work on the same constructs. If overlap is unavoidable, consider combining the work to be done on that construct and assign one person to the entire template.

Another side benefit to this approach is that it lends itself well to teaching XSLT coding to novices. When teaching XSLT concepts, it's useful to offer relatively small, self-contained exercises. If you're in the position to teach or mentor a co-worker, consider assigning them some of the easier changes to be made. Look for a wide variety of situations in which they will have to match a construct and then get the value of, copy, or restructure that construct or its child, parent, or sibling constructs. Due to the nature of these types of changes, it's likely that your student will have the opportunity to think through several possible cases and use conditional logic, which is always good practice.

Section 6. Create and test the XSLT conversion stylesheet

You should now have a set of tested XSLT templates that will implement the necessary XML instance document updates. You might have one template per stylesheet; or, by this time, you might have combined the templates into one or more common stylesheets. If you did the latter, you already have a head start on this step. Here's the process, which you've probably guessed by now:

1. Combine all of your templates into one stylesheet. Technically, the order of the templates doesn't matter; alphabetizing them makes it easier to find them within the stylesheet.
2. Test the stylesheet using XML instance documents that will cause your templates to fire and alter the output, as well as documents that will leave the instance documents unchanged. By this stage, you should already have a good list of sample instance documents due to the testing of individual templates.
3. Change your stylesheet to reflect any problems that testing exposes. The only problems you might expect to encounter at this stage are those caused by overlapping template code (templates that process the same

situations differently). This is usually the result of different people developing similar templates or from requirement updates since the original templates were coded and tested.

4. If you have the luxury of time and willing people, ask some of your XML instance document authors and editors to transform some of their representative content with the conversion stylesheet to confirm that the result is an expected one.

It's worth emphasizing that I've added no additional XSLT code to this stylesheet other than the stylesheet declaration information at the top, some comments, and a stylesheet closing tag at the bottom.

Listing 3. Example conversion stylesheet: All conversion templates combined

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Conversion stylesheet for updating V1-based xml documents to V2
-->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
exclude-result-prefixes="fo xsi xsl">
  <xsl:output method="xml" indent="no" omit-xml-declaration="yes"
    encoding="UTF-8"/>
  <!-- Change employee element as follows:
    1. Replace dept element with new dept attribute.
    2. Replace old dept numbers with new ones
    3. Delete location element.
    4. Add manager element.
  -->
  <xsl:template match="employee">
    <xsl:element name="employee">
      <!-- Add dept attribute; source from dept element. -->
      <xsl:attribute name="dept">
        <xsl:choose>
          <!-- Replace old dept numbers with new values -->
          <xsl:when test="dept='012'">
            <xsl:text>01</xsl:text>
          </xsl:when>
          <xsl:when test="dept='123'">
            <xsl:text>02</xsl:text>
          </xsl:when>
          <xsl:when test="dept='456'">
            <xsl:text>03</xsl:text>
          </xsl:when>
          <xsl:when test="dept='789'">
            <xsl:text>04</xsl:text>
          </xsl:when>
          <xsl:otherwise>
            <xsl:text>05</xsl:text>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:attribute>
      <xsl:for-each select="*">
        <xsl:choose>
          <!-- Delete dept and location elements. -->
          <xsl:when test="name()='dept' or name()='location'"/>
          <xsl:otherwise>
            <xsl:choose>
              <xsl:when test="name()='phone'">
```

```

        <!-- The match="phone" template will be called -->
        <xsl:apply-templates select="."/>
    </xsl:when>
    <xsl:otherwise>
        <xsl:element name="{name()}">
            <xsl:value-of select="."/>
        </xsl:element>
    </xsl:otherwise>
</xsl:choose>
</xsl:otherwise>
</xsl:choose>
</xsl:for-each>
<!-- Add manager element -->
<xsl:element name="manager">
    <xsl:choose>
        <xsl:when test="dept='012'">
            <xsl:text>Ms. Alpha</xsl:text>
        </xsl:when>
        <xsl:when test="dept='123'">
            <xsl:text>Mr. Bravo</xsl:text>
        </xsl:when>
        <xsl:when test="dept='456'">
            <xsl:text>Mr. Charlie</xsl:text>
        </xsl:when>
        <xsl:when test="dept='789'">
            <xsl:text>Ms. Delta</xsl:text>
        </xsl:when>
        <xsl:otherwise>
            <xsl:text>Ms. Parker</xsl:text>
        </xsl:otherwise>
    </xsl:choose>
</xsl:element>
</xsl:element>
</xsl:template>

    <!-- Change phone element as follows:
        Replace '123' area code with '333'.
    -->
<xsl:template match="phone">
    <!-- Change area code 123 to 333, but keep the exchange+number
-->
    <xsl:variable name="exchange-number">
        <xsl:value-of select="substring-after(.,'123')"/>
    </xsl:variable>
    <xsl:element name="phone">
        <xsl:value-of select="concat('333', $exchange-number)"/>
    </xsl:element>
</xsl:template>
</xsl:stylesheet>

```

Section 7. In conclusion

In Part 1 of this tutorial, I introduce the steps involved in updating many XML instance documents using XSLT conversion stylesheets and Apache Ant. You reviewed a method to determine the XML instance document updates. Lastly, you translated those update requirements into a conversion stylesheet composed of individual conversion templates that perform the XML updates.

In [Part 2](#), you'll install Apache Ant and Java SE. You'll create the XML-based instructions that Ant uses to transform existing XML instance documents into new ones (using our conversion stylesheet). For an added bonus, I'll cover XML file validation within the Ant process, as well as transforming the newly-updated XML files into HTML.

Sample schemas, XML instance documents, and a conversion stylesheet are available at [Downloads](#).

Also be sure to review [Resources](#) for more detailed information and links to XML topics, Ant, Java SE, and related developerWorks information on these topics.

Acknowledgments

First, thanks to our own [Doug Tidwell](#) who pioneered this process here at developerWorks several years ago. We've made some tweaks along the way, but the basic idea was his and we are far better off for it. Doug's [Web services contributions](#) and [XML contributions](#) to developerWorks have proven very popular.

Secondly, thanks go to the members of the developerWorks schema and stylesheet development team. Elizabeth, Frank, Jack, and Leah, your willingness to divide and conquer all of the schema and stylesheet work means more than you know. Janet and Kristin, we're grateful that you've joined the team; the list of to-do's never seems to dwindle, no matter how hard we try!

I also want to thank our management team. Your encouragement to think creatively, along with the flexibility you continue to allow, makes working for developerWorks the best job in IBM.

Downloads

Description	Name	Size	Download method
Sample Ant, XML, XSD, and XSLT files	wa-autoxml-file-updates.zip	9KB	HTTP

[Information about download methods](#)

Resources

Learn

- [Automate XML file updates, Part 2](#): Read Part 2 in this series.
- [developerWorks Web architecture zone](#): Expand your Web-building skills.
- [developerWorks XML zone](#): Visit the developerWorks XML zone to obtain a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.

Get products and technologies

- [Altova](#): Get the XML Spy editor.
- [The Apache Software Foundation](#): Retrieve Ant and the Xalan XSLT processor.
- [IBM developer kits](#): Get downloads and documentation for current releases of Java SE.
- [IDM Computer Solutions, Inc.](#): Acquire the Ultra-Edit text editor.
- [Sun Developer Network \(SDN\)](#): Download Java SE and get corresponding documentation.

Discuss

- [developerWorks XML forums](#): Communicate with other XML developers trying to solve the same problems you are.
- [developerWorks Community](#): Visit blogs, forums, podcasts, and wikis hosted by and for developers.

About the author

Tom Coppedge

Tom Coppedge has been a member of the developerWorks design team since the site was launched in 1999. Tom's focus includes XML & XSLT strategy, information architecture, and site design. He joined IBM in 1988 after receiving a degree in Information Systems & Operations Management from the University of North Carolina at Greensboro.

Trademarks

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.