Automate XML file updates, Part 2: Use Apache Ant and conversion stylesheets to update your XML

A methodology using XSLT, Apache Ant, and Java SE

Skill Level: Intermediate

Tom Coppedge (tcoppedg@us.ibm.com)
developerWorks Software Engineer
IBM

17 Aug 2006

This is the second 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 to install, configure, and run 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

Here's 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, by the needs of 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 a 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 (Go directly to Part 1 now.)

- 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

- 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
  - Transform the updated XML files 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 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. Apache Ant: Introduction and installation instructions

The Apache Software Foundation (see Resources) describes Ant as a Java-based build tool. If you've worked with *make*, another build tool, Ant's functions will be somewhat familiar to you. For the methodology described in this tutorial, Ant is a process controller and will control these steps for you:

1. Create working directories.
2. Determine which files to process.
3. Create new XML instance documents.
4. Validate the new XML instance documents against your new schema.
5. Transform the updated XML instance documents to HTML.

Ant gets its processing instructions from a simple XML configuration file that you create. No Java technology programming experience is required.

Download and install Ant

These instructions assume you're using Windows XP. The Apache Software Foundation states that Ant will run on "...Linux, commercial flavours of UNIX such as Solaris and HP-UX, Windows 9x and NT, OS/2 Warp, Novell Netware 6 and MacOS X". See the online Ant manual for more information.

1. Download Ant from the Apache site. You'll want the *.zip version for Windows machines.
2. Uncompress the ZIP file into a directory on your machine. These

instructions assume you'll save Ant version 1.6.5 in the `C:\Program Files` directory. The path to Ant would then be: `C:\Program Files\apache-ant-1.6.5`

3. Open the online manual stored in `C:\Program Files\apache-ant-1.6.5\docs\index.html`, and read the **System Requirements**, **Installing Ant**, and **Platform Specific Issues** sections. At the very least you'll have to:
   - Add the `C:\Program Files\apache-ant-1.6.5\bin` directory to your path
   - Create an environment variable for `ANT_HOME` with a value of `C:\Program Files\apache-ant-1.6.5`

This concludes the Ant installation. However, Ant requires Java™ SE (or EE), so install that next.

Section 4. Download and install Java SE

You can install J2SE™ Runtime Environment or J2SE Development Kit. The Runtime Environment is sufficient for the process described in this tutorial. J2EE™ will also work, but is not needed for this process.

You will transform XML files, so you also need an XSL transformer. If you install Java SE version 1.4 or later (Runtime or Development Kit), an XSL transformer is included. Otherwise, you need to install an XSL transformer, such as Xalan-Java, separately. For this tutorial, I assume you will download J2SE Runtime Environment 1.4.2 which comes with an XSL transformer.

Download information for non-IBM employees

The IBM developer kits page has downloads and documentation for current releases of Java SE for various operating systems, including the IBM 32-bit Runtime Environment for Windows.

Similarly, the Sun Developer Network (SDN) has Java SE downloads and documentation.

Download information for IBM employees

A legal agreement between IBM and Sun Microsystems, Inc. requires IBM employees to download Java specifications, reference implementations or test compatibility kits only from the Java Information Manager (JIM) IBM intranet site. IBM employees may not download them from the Sun Microsystems site, nor the Java Community Process (JCP) site.

However, if IBM employees need only the Runtime Environment (as is the case for this tutorial), and are not shipping Java technology functionality within products they're developing, they may download the IBM Java Runtime (version 1.4.2 SR5 for Windows, currently) from the IBM Standard Software Installer (ISSI) IBM intranet site. The ISSI installation process handles the necessary machine configuration steps for you.

Regardless of your download source, please read the system requirements and installation instructions carefully.

Section 5. Create a build.xml file for Ant

Ant refers to an XML configuration file that you create for its processing instructions. You may give any name to this file; by default, it looks for a file named `build.xml`. You can create a build.xml file that instructs Ant to:

1. Create working directories.
2. Determine which files to process.
3. Create new XML instance documents.
4. Validate the new XML instance documents against your new schema.
5. Transform the updated XML instance documents to HTML.

It's important to note that Ant is an extremely flexible tool and has a much wider variety of functions available than what I introduce here.

The documentation that comes with Ant is very good; it's stored in the `/docs` directory under the primary Ant directory on your machine. The information in the `/manual` directory will be most helpful as you begin coding your build.xml files. Augment the descriptions below with a review of their functions in the Ant manual.

How the build.xml file is structured

The build.xml file is composed of one or more property settings and a set of processing instructions. Each processing instruction is called a *target*, and is contained within a `target` element. A target can convey several pieces of information, such as:

- The target's name
- Description
- The name of the target that must be executed before this one
- Base, source, destination, and classpath directories associated with the target
- File names and file extension names associated with the base and destination files referenced by the target

...and many other optional values, depending on your use of Ant.

A sample build.xml file explained

First, review **Directory-based tasks** in your Ant manual before proceeding; it will help you understand the directory-related syntax in the examples below. When you're ready to proceed, refer to A sample build.xml file in a separate browser window while you read this section. I review each part of the sample build.xml file below.

Listing 1. The project element

```
<project name="xml-conversion" default="go" basedir=".">
```

The `project` element associates the build.xml file with a specific project. Its primary use is to tell Ant which target to start with if none is specified when Ant is run, and the directory, relative to where build.xml is located, from which the file paths in all targets are based. In the sample file, you tell Ant to start with the `go` target, and that all file paths are relative to the current (.) directory.

Listing 2. The description element

```
<description>Converts XML files from version 1 to version 2</description>
```

The `description` element is meant for humans to read. You should describe what your Ant process does here.

Listing 3. The property element

```
<!-- Holds the original source XML files. These aren't changed at all. -->
<property name="src-xml" location="v1xml"></property>

<!-- Holds XML files that have been converted to V2-->
<property name="converted-xml" location="v2xml"></property>

<!-- Holds HTML files transformed from V2-based XML files -->
<property name="html" location="V2html"></property>
```

You use `property` elements to define aliases for the directories Ant will use. The `name` attribute defines the alias, while the `location` attribute defines the actual file system directory path, relative to the base directory defined earlier in the `project` element. For example, you defined the alias `src-xml` for the actual directory named `v1xml`. Because you previously defined the base directory to be the current (.) directory, the one where the build.xml file resides, v1xml is, therefore, a subdirectory of the current directory.

Listing 4. The target element: init

```
<target name="init"
              description="Create the directory structure.">
   <!-- Create the time stamp -->
   <tstamp></tstamp>

   <!-- Create the directories for output XML and HTML files -->
   <mkdir dir="${converted-xml}"></mkdir>
   <mkdir dir="${html}"></mkdir>
</target>
```

You use this target element to create a time stamp (`<tstamp></tstamp>`) for the Ant job, and to create the file system directories where Ant will store the XML output from the conversion process (the `converted-xml` alias) and the HTML output from the transformation of the new XML into HTML (the `html` alias). Time stamps can be useful if you wish to redirect message or error output from Ant to a file instead of your screen (usually the default).

Listing 5. The target element: deleteUnnecessaryFiles

```
<target name="deleteUnnecessaryFiles"
              depends="init"
              description="Delete files we don't need.">
   <delete>
      <fileset dir="${src-xml}"
                  includes="**/*.*"
                  excludes="**/*.xml">
      </fileset>
   </delete>
</target>
```

You use this `target` element to ensure you're only processing XML files in the source directory. The `depends` attribute specifies that the init target must be

processed before this target. The `delete` element, which performs the actual file deletion(s), contains a `fileset` element. The `fileset` element is where you define:

- The directory from which files are deleted (the `dir` attribute). In our example, the value is the alias `src-xml`, which corresponds to the `v1xml` directory on the file system.
- Which files are to considered for the delete action (the `includes` attribute). In our example, the value is `**/*.*`, which means, "All directories under the `src-xml` directory" (`**`), and within them, all files (`*.*`).
- Of those files being considered, which files are excluded from the delete action (the `excludes` attribute). In our example, the value is `**/*.xml`, which means, "All directories under the `src-xml` directory" (`**`), and within them, all files with `xml` file extensions (`*.xml`).

Listing 6. The target element: convertToV2

```
<target name="convertToV2"
              depends="deleteUnnecessaryFiles"
              description="Apply the conversion stylesheet to all V1 XML files">
   <xslt basedir="${src-xml}"
            destdir="${converted-xml}"
            extension=".xml"
            style="convert-to-v2.xsl"
            includes="**/*.xml"
            excludesFile="../excludes.text">
   </xslt>
</target>
```

This target element is responsible for applying the conversion stylesheet. The stylesheet is applied to all the existing XML files specified in the `includes` and `excludesFile` attributes in order to transform them into updated XML files that conform to the new schema. The components of this target are:

- The `depends` attribute:
  The value `deleteUnnecessaryFiles` here means that the deleteUnnecessaryFiles target must be processed before this target.
- The `xslt` element:
  - The `basedir` attribute:
    The (alias) value `src-xml` means "look there for files to transform."
  - The `destdir` attribute:
    The (alias) value `converted-xml` means "store the transformed XML files here."
  - The `classpath` attribute:

Though not shown in the example, the classpath value specifies the location of the XSL transformer's .jar files. Refer to your Ant version's manual for more information on the possible values, or whether you need this attribute at all. Your version of Ant might scan your machine's classpath by default to look for and use the first TrAX-compliant (the Java Transformation API for XML) XSL processor it encounters, negating the need for this classpath attribute. For example, both Xalan 2 and Saxon are TrAX-compliant XSL processors.

- The `extension` attribute:
  The file extension `.xml` means "the transformed files should have the .xml file extension."
- The `style` attribute:
  The value of `convert-to-v2.xsl` means "transform the files using the stylesheet named convert-to-v2.xsl."
- The `includes` attribute:
  You've seen this before, and it means the same thing in this context: Include all the files with .xml extensions in all the directories within the directory named in the `basedir` attribute.
- The `excludesFile` attribute:
  With this attribute, Ant gives you the capability to list, in a separate file, the names of individual files (wildcards accepted) that are to be excluded from processing, even if they meet the criteria established by the `includes` attribute. Each line of the file expresses a pattern that should identify one or more files. This is very helpful if you know you have files that don't meet certain criteria necessary to process them further, such as: down-level, pre-production, archived

Listing 7. The target element: validateV2

```
<target name="validateXML"
    depends="convertToV2"
    description="Validates all newly-updated XML files against employee2.xsd"
    <xmlvalidate failonerror="yes">
      <fileset dir="${converted-xml}"
                 includes="**/*.xml">
      </fileset>
      <attribute name="http://apache.org/xml/features/validation/schema" value="true" />
      <attribute name="http://xml.org/sax/features/namespaces" value="true" />
      <property
      name="http://apache.org/xml/properties/schema/external-noNamespaceSchemaLocation"
      value="E:\developerworks\library\tom\toms-xml-tutorial\employee2.xsd"/>
    </xmlvalidate>
</target>
```

This target element validates the newly-updated XML files against the new schema. The components of this target you haven't seen previously are:

- The `xmlvalidate` element:
  This is the element that does the work. Within it is a `failonerror` attribute, and two `attribute` elements (an unfortunately confusing name for an element) you haven't see before:
  - The `failonerror` attribute:
    The attribute value of `true` means "Stop the Ant build if an invalid file is found." If set to `no`, the build will continue, even if invalid documents are found.
  - The first `attribute` element tells Ant to use its built-in JAXP parser to perform the validation.
  - Since JAXP's default setting is not to be namespace-aware, you need to specify the second `attribute` element, which essentially turns on namespace awareness.
- The `property` element:
  Here you set a property to tell Ant where the schema is located. You do this because the output XML files in this project will not have a schema declaration within them. You don't need this property otherwise.

Listing 8. The target element: transformToHtml

```
<target name="transformToHTML"
              depends="convertToV2"
              description="Transforms all V2 xml documents into HTML.">
   <xslt basedir="${converted-xml}"
   destdir="${html}"
   extension=".html"
   style="transform-to-html.xsl"
   includes="**/*.xml"
   excludesFile="../excludes.text">
   </xslt>
   <fixcrlf srcdir="${html}"
              destdir="${html}"
              tab="remove"
              tablength="2"
              eol="crlf"
              includes="**/*.html">
   </fixcrlf>
</target>
```

This target element is responsible for applying a stylesheet that transforms the updated XML files into HTML. The only new attribute of this target is `fixcrlf`, which will convert, in the HTML files produced:

- Any tab characters to (2) spaces
- Any eol characters to crlf (carriage return + line feed)

...and save the HTML back into the same (alias `html`) directory.

Listing 9. The target element: go

```
<target name="go" depends="transformToHTML"></target>
```

This is the target element, named as the default target, Ant is to start its execution with if no other target is specified. Refer back to the description of the `<project>` element, and see where you named `go` as the default. The go target depends on the transformToHTML target, which has its own dependency, and so on.

Listing 10. The target element: cleanup

```
<target name="cleanup"
              description="Delete output directories and their contents">
   <!-- Delete the created directory trees -->
   <delete dir="${converted-xml}"></delete>
   <delete dir="${html}"></delete>
</target>
```

This target is useful if you need to delete both output directories (and their contents) with one command. You'll probably run this target many times over the course of a project when you know you have incorrect or incomplete output for whatever reason.

Next, I review how Ant is run from the command line and look at some output.

Section 6. Run Ant to create new XML and HTML files

Assuming your build file is named `build.xml`, to run Ant from the command line, simply change directories to the directory where your build file is saved, type `ant`, and press **Enter**.

If you choose to name your build file something other than `build.xml`, like `build-tom.xml`, then you'd type:

Listing 11. Refer to a non-standard build file name

```
ant -buildfile build-tom.xml
```

Be sure to read the Ant manual for more command line options, as well as options for running Ant from within Java programs.

Ant output

The following figure displays what Ant 1.6.5 running on Windows XP returns during

and after execution of a simple excercise to validate four XML documents.

Listing 12. Sample Ant output

```
Buildfile: validateV2.xml

ValidateXML:
[xmlvalidate] 4 file(s) have been successfully validated.

BUILD SUCCESSFUL
Total time: 2 seconds
```

To get this output, I redirected the output to a file named ant-output.txt using the following command at a DOS prompt:

Listing 13. Redirect Ant output to a file

```
ant -buildfile validateV2.xml > ant-output.txt
```

Refer to this sample Ant output for an example of the messages you receive if you use Ant to process the sample files included with this tutorial.

Section 7. In conclusion

In this second part of this 2-part tutorial series, I introduced and you installed Apache Ant, along with Java SE. You created the XML-based instructions that Ant uses to transform existing XML instance documents into new ones (using your conversion stylesheet). You also incorporated XML file validation into the Ant build process and transformed the newly-updated XML files into HTML.

A sample Ant build.xml file is 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 been very popular.

Secondly, I have to thank 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 1: Read Part 1 of 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: Acquire Ant and the Xalan XSLT processor.
- IBM developer kits: Download current releases of Java SE and view documentation.
- IDM Computer Solutions, Inc.: Get the Ultra-Edit text editor.
- Sun Developer Network (SDN) Downlaod Java SE and view the 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.