Working with XML

The Java API for Xml Parsing (JAXP) Tutorial

by Eric Armstrong

[Version 1.1, Update 31 -- 21 Aug 2001]

This tutorial covers the following topics:

Part I: Understanding XML and the Java XML APIs explains the basics of XML and gives you a guide to the acronyms associated with it. It also provides an overview of the Java^{TM} XML APIs you can use to manipulate XML-based data, including the Java API for XML Parsing ((JAXP). To focus on XML with a minimum of programming, follow The XML Thread, below.

Part II: Serial Access with the Simple API for XML (SAX) tells you how to read an XML file sequentially, and walks you through the callbacks the parser makes to event-handling methods you supply.

Part III: XML and the Document Object Model (DOM) explains the structure of DOM, shows how to use it in a JTree, and shows how to create a hierarchy of objects from an XML document so you can randomly access it and modify its contents. This is also the API you use to write an XML file after creating a tree of objects in memory.

Part IV: Using XSLT shows how the XSL transformation package can be used to write out a DOM as XML, convert arbitrary data to XML by creating a SAX parser, and convert XML data into a different format.

Additional Information contains a description of the character encoding schemes used in the Java platform and pointers to any other information that is relevant to, but outside the scope of, this tutorial.

The XML Thread

Scattered throughout the tutorial there are a number of sections devoted more to explaining the basics of XML than to programming exercises. They are listed here so as to form an XML thread you can follow without covering the entire programming tutorial:

Top Contents Index Glossary

Part I. Understanding XML and the Java XML APIs

This section describes the Extensible Markup Language (XML), its related specifications, and the APIs for manipulating XML files. It contains the following files:

What You'll Learn

This section of the tutorial covers the following topics:

1. A Quick Introduction to XML shows you how an XML file is structured and gives you some ideas about how to use XML.
2. XML and Related Specs: Digesting the Alphabet Soup helps you wade through the acronyms surrounding the XML standard.
3. An Overview of the APIs gives you a high-level view of the JAXP and associated APIs.
4. Designing an XML Data Structure gives you design tips you can use when setting up an XML data structure.

1. A Quick Introduction to XML

This page covers the basics of XML. The goal is to give you just enough information to get started, so you understand what XML is all about. (You'll learn about XML in later sections of the tutorial.) We then outline the major features that make XML great for information storage and interchange, and give you a general idea of how XML can be used. This section of the tutorial covers:

- What Is XML?
- Why Is XML Important?
- How Can You Use XML?

Link Summary

Local Links

- XML and Related Specs
- Designing an XML Data Structure
- RDF
- XSL

External Links

- XML FAQ
- XML Info and Recommended Reading
- SGML/XML Web Page
- Scientific American article

Glossary Terms

attributes, declaration, DTD, element, entity, prolog, tag, well-formed

What Is XML?

XML is a text-based markup language that is fast becoming the standard for data interchange on the Web. As with HTML, you identify data using tags (identifiers enclosed in angle brackets, like this: <...>). Collectively, the tags are known as "markup".

But unlike HTML, XML tags *identify* the data, rather than specifying how to display it. Where an HTML tag says something like "display this data in bold font" (`<b>...</b>`), an XML tag acts like a field name in your program. It puts a label on a piece of data that identifies it (for example: `<message>...</message>`).

Note:
Since identifying the data gives you some sense of what *means* (how to interpret it, what you should do with it), XML is sometimes described as a mechanism for specifying the *semantics* (meaning) of the data.

In the same way that you define the field names for a data structure, you are free to use any XML tags that make sense for a given application. Naturally, though, for multiple applications to use the same XML data, they have to agree on the tag names they intend to use.

Here is an example of some XML data you might use for a messaging application:

```
<message>
    <to>you@yourAddress.com</to>
    <from>me@myAddress.com</from>
    <subject>XML Is Really Cool</subject>
    <text>
       How many ways is XML cool? Let me count the ways...
    </text>
</message>
```

> **Note:** Throughout this tutorial, we use boldface text to highlight things we want to bring to your attention. XML does not require anything to be in bold!

The tags in this example identify the message as a whole, the destination and sender addresses, the subject, and the text of the message. As in HTML, the `<to>` tag has a matching end tag: `</to>`. The data between the tag and and its matching end tag defines an element of the XML data. Note, too, that the content of the `<to>` tag is entirely contained within the scope of the `<message>..</message>` tag. It is this ability for one tag to contain others that gives XML its ability to represent hierarchical data structures

Once again, as with HTML, whitespace is essentially irrelevant, so you can format the data for readability and yet still process it easily with a program. Unlike HTML, however, in XML you could easily search a data set for messages containing "cool" in the subject, because the XML tags identify the content of the data, rather than specifying its representation.

Tags and Attributes

Tags can also contain attributes -- additional information included as part of the tag itself, within the tag's angle brackets. The following example shows an email message structure that uses attributes for the "to", "from", and "subject" fields:

```
<message to="you@yourAddress.com" from="me@myAddress.com"
         subject="XML Is Really Cool">
```

```
    <text>
       How many ways is XML cool? Let me count the ways...
    </text>
</message>
```

As in HTML, the attribute name is followed by an equal sign and the attribute value, and multiple attributes are separated by spaces. Unlike HTML, however, in XML commas between attributes are not ignored -- if present, they generate an error.

Since you could design a data structure like `<message>` equally well using either attributes or tags, it can take a considerable amount of thought to figure out which design is best for your purposes. The last part of this tutorial, Designing an XML Data Structure, includes ideas to help you decide when to use attributes and when to use tags.

Empty Tags

One really big difference between XML and HTML is that an XML document is always constrained to be well formed. There are several rules that determine when a document is well-formed, but one of the most important is that every tag has a closing tag. So, in XML, the `</to>` tag is not optional. The `<to>` element is never terminated by any tag other than `</to>`.

> **Note:** Another important aspect of a well-formed document is that all tags are completely nested. So you can have `<message>..<to>..</to>..</message>`, but never `<message>..<to>..</message>..</to>`. A complete list of requirements is contained in the list of XML Frequently Asked Questions (FAQ) at `http://www.ucc.ie/xml/#FAQ-VALIDWF`. (This FAQ is on the w3c "Recommended Reading" list at `http://www.w3.org/XML/`.)

Sometimes, though, it makes sense to have a tag that stands by itself. For example, you might want to add a "flag" tag that marks message as important. A tag like that doesn't enclose any content, so it's known as an "empty tag". You can create an empty tag by ending it with `/>` instead of `>`. For example, the following message contains such a tag:

```
<message to="you@yourAddress.com" from="me@myAddress.com"
         subject="XML Is Really Cool">
    <flag/>
    <text>
       How many ways is XML cool? Let me count the ways...
    </text>
```

```
</message>
```

Note: The empty tag saves you from having to code `<flag></flag>` in order to have a well-formed document. You can control which tags are allowed to be empty by creating a Document Type Definition, or DTD. We'll talk about that in a few moments. If there is no DTD, then the document can contain any kinds of tags you want, as long as the document is well-formed.

Comments in XML Files

XML comments look just like HTML comments:

```
<message to="you@yourAddress.com" from="me@myAddress.com"
         subject="XML Is Really Cool">
    <!-- This is a comment -->
    <text>
       How many ways is XML cool? Let me count the ways...
    </text>
</message>
```

The XML Prolog

To complete this journeyman's introduction to XML, note that an XML file always starts with a prolog. The minimal prolog contains a declaration that identifies the document as an XML document, like this:

```
<?xml version="1.0"?>
```

The declaration may also contain additional information, like this:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

The XML declaration is essentially the same as the HTML header, `<html>`, except that it uses `<?..?>` and it may contain the following attributes:

version
: Identifies the version of the XML markup language used in the data. This attribute is not optional.

encoding
: Identifies the character set used to encode the data. "ISO-8859-1" is "Latin-1" the Western European and English language character set. (The default is compressed

Unicode: UTF-8.)

standalone

Tells whether or not this document references an external entity or an external data type specification (see below). If there are no external references, then "yes" is appropriate

The prolog can also contain definitions of entities (items that are inserted when you reference them from within the document) and specifications that tell which tags are valid in the document, both declared in a Document Type Definition (DTD) that can be defined directly within the prolog, as well as with pointers to external specification files. But those are the subject of later tutorials. For more information on these and many other aspects of XML, see the Recommended Reading list of the w3c XML page at `http://www.w3.org/XML/`.

> **Note:** The declaration is actually optional. But it's a good idea to include it whenever you create an XML file. The declaration should have the version number, at a minimum, and ideally the encoding as well. That standard simplifies things if the XML standard is extended in the future, and if the data ever needs to be localized for different geographical regions.

Everything that comes after the XML prolog constitutes the document's *content*.

Processing Instructions

An XML file can also contain *processing instructions* that give commands or information to an application that is processing the XML data. Processing instructions have the following format:

```
  <?target instructions?>
```

where the *target* is the name of the application that is expected to do the processing, and *instructions* is a string of characters that embodies the information or commands for the application to process.

Since the instructions are application specific, an XML file could have multiple processing instructions that tell different applications to do similar things, though in different ways. The XML file for a slideshow, for example, could have processing instructions that let the speaker specify a technical or executive-level version of the presentation. If multiple presentation programs were used, the program might need multiple versions of the processing instructions (although it would be nicer if such applications recognized standard instructions).

> **Note:** The target name "xml" (in any combination of upper or lowercase letters) is reserved for XML standards. In one sense, the declaration is a processing instruction that fits that standard. (However, when you're working with the parser later, you'll see that the method for handling processing instructions never sees the declaration.)

Why Is XML Important?

There are a number of reasons for XML's surging acceptance. This section lists a few of the most prominent.

Plain Text

Since XML is not a binary format, you can create and edit files with anything from a standard text editor to a visual development environment. That makes it easy to debug your programs, and makes it useful for storing small amounts of data. At the other end of the spectrum, an XML front end to a database makes it possible to efficiently store large amounts of XML data as well. So XML provides scalability for anything from small configuration files to a company-wide data repository.

Data Identification

XML tells you what kind of data you have, not how to display it. Because the markup tags identify the information and break up the data into parts, an email program can process it, a search program can look for messages sent to particular people, and an address book can extract the address information from the rest of the message. In short, because the different parts of the information have been identified, they can be used in different ways by different applications.

Stylability

When display is important, the stylesheet standard, XSL, lets you dictate how to portray the data. For example, the stylesheet for:

```
<to>you@yourAddress.com</to>
```

can say:

1. Start a new line.
2. Display "To:" in bold, followed by a space
3. Display the destination data.

Which produces:

```
To: you@yourAddress
```

Of course, you could have done the same thing in HTML, but you wouldn't be able to process the data with search programs and address-extraction programs and the like. More importantly, since XML is inherently style-free, you can use a completely different stylesheet to produce output in postscript, TEX, PDF, or some new format that hasn't even been invented yet. That flexibility amounts to what one author described as "future-proofing" your information. The XML documents you author today can be used in future document-delivery systems that haven't even been imagined yet.

Inline Reusabiliy

One of the nicer aspects of XML documents is that they can be composed from separate entities. You can do that with HTML, but only by linking to other documents. Unlike HTML, XML entities can be included "in line" in a document. The included sections look like a normal part of the document -- you can search the whole document at one time or download it in one piece. That lets you modularize your documents without resorting to links. You can single-source a section so that an edit to it is reflected everywhere the section is used, and yet a document composed from such pieces looks for all the world like a one-piece document.

Linkability

Thanks to HTML, the ability to define links between documents is now regarded as a necessity. The next section of this tutorial, XML and Related Specs, discusses the link-specification initiative. This initiative lets you define two-way links, multiple-target links, "expanding" links (where clicking a link causes the targeted information to appear inline), and links between two existing documents that are defined in a third.

Easily Processed

As mentioned earlier, regular and consistent notation makes it easier to build a program to process XML data. For example, in HTML a `<dt>` tag can be delimited by `</dt>`, another `<dt>`, `<dd>`, or `</dl>`. That makes for some difficult programming. But in XML, the `<dt>` tag must always have a `</dt>` terminator, or else it will be defined as a `<dt/>` tag. That restriction is a critical part of the constraints that make an XML document well-formed. (Otherwise, the XML parser won't be able to read the data.) And since XML is a vendor-neutral standard, you can choose among several XML parsers, any one of which takes the work out of processing XML data.

Hierarchical

Finally, XML documents benefit from their hierarchical structure. Hierarchical document structures are, in general, faster to access because you can drill down to the part you need, like stepping through a table of contents. They are also easier to rearrange, because each piece is delimited. In a document, for example, you could move a heading to a new location and drag everything under it along with the heading, instead of having to page down to make a selection, cut, and then paste the selection into a new location.

How Can You Use XML?

There are several basic ways to make use of XML:

- Traditional data processing, where XML encodes the data for a program to process
- Document-driven programming, where XML documents are containers that build interfaces and applications from existing components
- Archiving -- the foundation for document-driven programming, where the customized version of a component is saved (archived) so it can be used later
- Binding, where the DTD or schema that defines an XML data structure is used to automatically generate a significant portion of the application that will eventually process that data

Traditional Data Processing

XML is fast becoming the data representation of choice for the Web. It's terrific when used in conjunction with network-centric Java-platform programs that send and retrieve information. So a client/server application, for example, could transmit XML-encoded data back and forth between the client and the server.

In the future, XML is potentially the answer for data interchange in all sorts of transactions, as long as both sides agree on the markup to use. (For example, should an email program expect to see tags named `<FIRST>` and `<LAST>`, or `<FIRSTNAME>` and `<LASTNAME>`?) The need for common standards will generate a lot of industry-specific standardization efforts in the years ahead. In the meantime, mechanisms that let you "translate" the tags in an XML document will be important. Such mechanisms include projects like the RDF initiative, which defines "meta tags", and the XSL specification, which lets you translate XML tags into other XML tags.

Document-Driven Programming (DDP)

The newest approach to using XML is to construct a document that describes how an application page should look. The document, rather than simply being displayed, consists of references to user interface components and business-logic components that are "hooked together" to create an application on the fly.

Of course, it makes sense to utilize the Java platform for such components. Both Java Beans™ for interfaces and Enterprise Java Beans™ for business logic can be used to construct such applications. Although none of the efforts undertaken so far are ready for commercial use, much preliminary work has already been done.

> **Note:** The Java programming language is also excellent for writing XML-processing tools that are as portable as XML. Several Visual XML editors have been written for the Java platform. For a listing of editors, processing tools, and other XML resources, see the "Software" section of Robin Cover's SGML/XML Web Page.

Binding

Once you have defined the structure of XML data using either a DTD or the one of the schema standards, a large part of the processing you need to do has already been defined. For example, if the schema says that the text data in a `<date>` element must follow one of the recognized date formats, then one aspect of the validation criteria for the data has been defined -- it only remains to write the code. Although a DTD specification cannot go the same level of detail, a DTD (like a schema) provides a grammar that tells which data structures can occur, in what sequences. That specification tells you how to write the high-level code that processes the data elements.

But when the data structure (and possibly format) is fully specified, the code you need to process it can just as easily be generated automatically. That process is known as *binding* -- creating classes that recognize and process different data elements by processing the specification that defines those elements. As time goes on, you should find that you are using the data specification to generate significant chunks of code, so you can focus on the programming that is unique to your application.

Archiving

The Holy Grail of programming is the construction of reusable, modular components. Ideally, you'd like to take them off the shelf, customize them, and plug them together to construct an application, with a bare minimum of additional coding and additional compilation.

The basic mechanism for saving information is called *archiving*. You archive a component by writing it to an output stream in a form that you can reuse later. You can then read it in and instantiate it using its saved parameters. (For example, if you saved a table component, its parameters might be the number of rows and columns to display.) Archived components can also be shuffled around the Web and used in a variety of ways.

When components are archived in binary form, however, there are some limitations on the kinds of changes you can make to the underlying classes if you want to retain compatibility with previously saved versions. If you could modify the archived version to reflect the change, that would solve the problem. But that's hard to do with a binary object. Such considerations have prompted a number of investigations into using XML for archiving. But if an object's state were archived in text form using XML, then anything and everything in it could be changed as easily as you can say, "search and replace".

XML's text-based format could also make it easier to transfer objects between applications written in different languages. For all of these reasons, XML-based archiving is likely to become an important force in the not-too-distant future.

Summary

XML is pretty simple, and very flexible. It has many uses yet to be discovered -- we are just beginning to scratch the surface of its potential. It is the foundation for a great many standards yet to come, providing a common language that different computer systems can use to exchange data with one another. As each industry-group comes up with standards for what they want to say, computers will begin to link to each other in ways previously unimaginable.

For more information on the background and motivation of XML, see this great article in Scientific American at `http://www.sciam.com/1999/0599issue/0599bosak.html`.

Top Contents Index Glossary

2. XML and Related Specs: Digesting the Alphabet Soup

Now that you have a basic understanding of XML, it makes sense to get a high-level overview of the various XML-related acronyms and what they mean. There is a lot of work going on around XML, so there is a lot to learn.

The current APIs for accessing XML documents either serially or in random access mode are, respectively, SAX and DOM. The specifications for ensuring the validity of XML documents are DTD (the original mechanism, defined as part of the XML specification) and various schema proposals (newer mechanisms that use XML syntax to do the job of describing validation criteria).

Other future standards that are nearing completion include the XSL standard -- a mechanism for setting up translations of XML documents (for example to HTML or other XML) and for dictating how the document is rendered. The transformation part of that standard, XSLT, is completed and covered in this tutorial. Another effort nearing completion is the XML Link Language specification (XLL), which enables links between XML documents.

Those are the major initiatives you will want to be familiar with. This section also surveys a number of other interesting proposals, including the HTML-lookalike standard, XHTML, and the meta-standard for describing the information an XML document contains, RDF. There are also standards efforts that aim to extend XML, including XLink, and XPointer.

Finally, there are a number of interesting standards and standards-proposals that build on XML, including Synchronized Multimedia Integration Language (SMIL), Mathematical Markup Language (MathML), Scalable Vector Graphics (SVG), and DrawML, as well as a number of eCommerce standards.

The remainder of this section gives you a more detailed description of these initiatives. To help keep things straight, it's divided into:

- Basic Standards
- Schema Standards
- Linking and Presentation Standards
- Knowledge Standards
- Standards that Build on XML

Skim the terms once, so you know what's here, and keep a copy of this document handy so you can refer to it whenever you see one of these terms in something you're reading. Pretty soon, you'll have them all committed to memory, and you'll be at least "conversant" with XML!

Link Summary

Local Links

- Defining a Document Type
- DOM: Manipulating Document Contents
- SAX: Serial Access with the Simple API
- Using XSLT

External Links

- Basic Standards
  - XML & DTD
  - Namespaces
  - XSL
- Schema Standards
  - RELAX
  - Schematron
  - SOX
  - TREX
  - XML Schema (Structures)
  - XML Schema (Datatypes)
- Linking and Presentation Standards
  - XML Linking
  - XHTML
- Knowledge Standards
  - RDF
  - RDF Schema
  - Topic Maps and the Web
  - XML Topic Maps
  - W3C Semantic Web

Standards that Build on XML
  - Extended Document Standards
    - DrawML
    - MathML
    - SMIL
    - SVG
  - eCommerce Standards
    - ICE
    - ebXML

Basic Standards

These are the basic standards you need to be familiar with. They come up in pretty much any discussion of XML.

- cXML
- CBL

Glossary Terms

DTD, entity, prolog

SAX
Simple API for XML

This API was actually a product of collaboration on the XML-DEV mailing list, rather than a product of the W3C. It's included here because it has the same "final" characteristics as a W3C recommendation.

You can also think of this standard as the "serial access" protocol for XML. This is the fast-to-execute mechanism you would use to read and write XML data in a server, for example. This is also called an event-driven protocol, because the technique is to register your handler with a SAX parser, after which the parser invokes your callback methods whenever it sees a new XML tag (or encounters an error, or wants to tell you anything else).

For more information on the SAX protocol, see Serial Access with the Simple API for XML.

DOM
Document Object Model

The Document Object Model protocol converts an XML document into a collection of objects in your program. You can then manipulate the object model in any way that makes sense. This mechanism is also known as the "random access" protocol, because you can visit any part of the data at any time. You can then modify the data, remove it, or insert new data. For more information on the DOM specification, see Manipulating Document Contents with the Document Object Model.

DTD
Document Type Definition

The DTD specification is actually part of the XML specification, rather than a separate entity. On the other hand, it is optional -- you can write an XML document without it. And there are a number of schema proposals that offer more flexible alternatives. So it is treated here as though it were a separate specification.

A DTD specifies the kinds of tags that can be included in your XML document, and the valid arrangements of those tags. You can use the DTD to make sure you don't create an invalid XML structure. You can also use it to make sure that the XML structure you are reading (or that got sent over the net) is indeed valid.

Unfortunately, it is difficult to specify a DTD for a complex document in such a way that it prevents all invalid combinations and allows all the valid ones. So constructing a DTD is something of an art. The DTD can exist at the front of the document, as part of the prolog. It can also exist as a separate entity, or it can be split between the document prolog and one or more additional entities.

However, while the DTD mechanism was the first method defined for specifying valid document structure, it was not the last. Several newer schema specifications have been devised. You'll learn about those momentarily.

For more information, see Defining a Document Type.

Namespaces

The namespace standard lets you write an XML document that uses two or more sets of XML tags in modular fashion. Suppose for example that you created an XML-based parts list that uses XML descriptions of parts supplied by other manufacturers (online!). The "price" data supplied by the subcomponents would be amounts you want to total up, while the "price" data for the structure as a whole would be something you want to display. The namespace specification defines mechanisms for qualifying the names so as to eliminate ambiguity. That lets you write programs that use information from other sources and do the right things with it.

The latest information on namespaces can be found at http://www.w3.org/TR/REC-xml-names.

XSL
Extensible Stylesheet Language

The XML standard specifies how to identify data, not how to display it. HTML, on the other hand, told how things should be displayed without identifying what they were. The XSL standard has two parts, XSLT (the transformation standard, described next) and XSL-FO (the part that covers *formatting objects*, also known as *flow objects*). XSL-FO gives you the ability to define multiple areas on a page and then link them together. When a text stream is directed at the collection, it fills the first area and then "flows" into the second when the first area is filled. Such objects are used by newsletters, catalogs, and periodical publications.

The latest W3C work on XSL is at http://www.w3.org/TR/WD-xsl.

XSLT (+XPATH)
Extensible Stylesheet Language for Transformations

The XSLT transformation standard is essentially a translation mechanism that lets you specify what to convert an XML tag into so that it can be displayed -- for example, in HTML. Different XSL formats can then be used to display the same data in different ways, for different uses. (The XPATH standard is an addressing mechanism that you use when constructing transformation instructions, in order to specify the parts of the XML structure you want to transform.)

For more information, see Using XSLT.

Schema Standards

A DTD makes it possible to validate the structure of relatively simple XML documents, but that's as far as it goes.

A DTD can't restrict the content of elements, and it can't specify complex relationships. For example, it is impossible to specify with a DTD that a <heading> for a <book> must have both a <title> and an <author>, while a <heading> for a <chapter> only needs a <title>. In a DTD, once you only get to specify the structure of the <heading> element one time. There is no context-sensitivity.

This issue stems from the fact that a DTD specification is not hierarchical. For a mailing address that contained several "parsed character data" (PCDATA) elements, for example, the DTD might look something like this:

```
<!ELEMENT mailAddress (name, address, zipcode)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT zipcode (#PCDATA)>
```

As you can see, the specifications are linear. That fact forces you to come up with new names for similar elements in different settings. So if you wanted to add another "name" element to the DTD that contained the <firstName>, <middleInitial>, and <lastName>, then you would have to come up with another identifier. You could not simply call it "name" without conflicting with the <name> element defined for use in a <mailAddress>.

Another problem with the nonhierarchical nature of DTD specifications is that it is not clear what comments are meant to explain. A comment at the top like `<!-- Address used for mailing via the postal system -->` would apply to all of the elements that constitute a mailing address. But a comment like `<!-- Addressee -->` would apply to the `name` element only. On the other hand, a comment like `<!-- A 5-digit string -->` would apply specifically to the `#PCDATA` part of the `zipcode` element, to describe the valid formats. Finally, DTDs do not allow you to formally specify field-validation criteria, such as the 5-digit (or 5 and 4) limitation for the `zipcode` field.

Finally, a DTD uses syntax which substantially different from XML, so it can't be processed with a standard XML parser. That means you can't read a DTD into a DOM, for example, modify it, and then write it back out again.

To remedy these shortcomings, a number of proposals have been made for a more database-like, hierarchical "schema" that specifies validation criteria. The major proposals are shown below.

XML Schema

A large, complex standard that has two parts. One part specifies structure relationships. (This is the largest and most complex part.) The other part specifies mechanisms for validating the content of XML elements by specifying a (potentially very sophisticated) *datatype* for each element. The good news is that XML Schema for Structures lets you specify any kind of relationship you can conceive of. The bad news is that it takes a lot of work to implement, and it takes a bit of learning to use. Most of the alternatives provide for simpler structure definitions, while incorporating the XML Schema datatype standard.

For more information on the XML Schema proposal, see the W3C specs XML Schema (Structures) and XML Schema (Datatypes).

RELAX
Regular Language description for XML

Simpler than XML Structure Schema, RELAX uses XML syntax to express the structure relationships that are present in a DTD, and adds the XML Datatype Schema mechanisms, as well. Includes a DTD to RELAX converter.

For more information on Relax, see `http://www.xml.gr.jp/relax/`.

SOX
Schema for Object-oriented XML

SOX is a schema proposal that includes extensible data types, namespaces, and embedded documentation.

For more information on SOX, see `http://www.w3.org/TR/NOTE-SOX`.

TREX
Tree Regular Expressions for XM

A means of expressing validation criteria by describing a *pattern* for the structure and content of an XML document. Includes a RELAX to TREX converter.

For more information on TREX, see `http://www.thaiopensource.com/trex/`.

Schematron
Schema for Object-oriented XML

An assertion-based schema mechanism that allows for sophisticated validation.

For more information on Schematron, see `http://www.ascc.net/xml/resource/schematron/schematron.html`.

Linking and Presentation Standards

Arguably the two greatest benefits provided by HTML were the ability to link between documents, and the ability to create simple formatted documents (and, eventually, very complex formatted documents). The following standards aim at preserving the benefits of HTML in the XML arena, and to adding additional functionality, as well.

XML Linking

These specifications provide a variety of powerful linking mechanisms, and are sure to have a big impact on how XML documents are used.

XLink: The XLink protocol is a proposed specification to handle links between XML documents. This specification allows for some pretty sophisticated linking, including two-way links, links to multiple documents, "expanding" links that insert the linked information into your document rather than replacing your document with a new page, links between two documents that are created in a third, independent document, and indirect links (so you can point to an "address book" rather than directly to the target document -- updating the address book then automatically changes any links that use it).

XML Base: This standard defines an attribute for XML documents that defines a "base" address, that is used when evaluating a relative address specified in the document. (So, for example, a simple file name would be found in the base-address directory.)

XPointer: In general, the XLink specification targets a document or document-segment using its ID. The XPointer specification defines mechanisms for "addressing into the internal structures of XML documents", without requiring the author of the document to have defined an ID for that segment. To quote the spec, it provides for "reference to elements, character strings, and other parts of XML documents, whether or not they bear an explicit ID attribute".

For more information on the XML Linking standards, see http://www.w3.org/XML/Linking.

XHTML

The XHTML specification is a way of making XML documents that look and act like HTML documents. Since an XML document can contain any tags you care to define, why not define a set of tags that look like HTML? That's the thinking behind the XHTML specification, at any rate. The result of this specification is a document that can be displayed in browsers and also treated as XML data. The data may not be quite as identifiable as "pure" XML, but it will be a heck of a lot easier to manipulate than standard HTML, because XML specifies a good deal more regularity and consistency.

For example, every tag in a well-formed XML document must either have an end-tag associated with it or it must end in `/>`. So you might see `<p>...</p>`, or you might see `<p/>`, but you will never see `<p>` standing by itself. The upshot of that requirement is that you never have to program for the weird kinds of cases you see in HTML where, for example, a `<dt>` tag might be terminated by `</DT>`, by another `<DT>`, by `<dd>`, or by `</dl>`. That makes it a lot easier to write code!

The XHTML specification is a reformulation of HTML 4.0 into XML. The latest information is at http://www.w3.org/TR/xhtml1.

Knowledge Standards

When you start looking down the road five or six years, and visualize how the information on the web will begin to turn into one huge knowledge base (the "semantic web"). For the latest on the semantic web, visit http://www.w3.org/2001/sw/. In the meantime, here are the fundamental standards you'll want to know about:

RDF
Resource Description Framework

RDF is a proposed standard for defining data about data. Used in conjunction with the XHTML specification, for example, or with HTML pages, RDF could be used to describe the content of the pages. For example, if your browser stored your ID information as `FIRSTNAME`, `LASTNAME`, and `EMAIL`, an RDF description could make it possible to transfer data to an application that wanted `NAME` and `EMAILADDRESS`. Just think: One day you may not need to type your name and address at every web site you visit!

For the latest information on RDF, see http://www.w3.org/TR/REC-rdf-syntax.

RDF Schema

The RDF Schema proposal allows the specification of consistency rules and additional information that describe how the statements in a Resource Description Framework (RDF) should be interpreted.

For more information on the RDF Schema recommendation, see `http://www.w3.org/TR/rdf-schema`.

XTM
XML Topic Maps

In many ways a simpler, more readily usable knowledge-representation than RDF, the topic maps standard is one worth watching. So far, RDF is the W3C standard for knowledge representation, but topic maps could possibly become the "developer's choice" among knowledge representation standards.

For more information on XML Topic Maps, http://www.topicmaps.org/xtm/index.html. For information on topic maps and the web, see http://www.topicmaps.org/.

Standards That Build on XML

The following standards and proposals build on XML. Since XML is basically a language-definition tool, these specifications use it to define standardized languages for specialized purposes.

Extended Document Standards

These standards define mechanisms for producing extremely complex documents -- books, journals, magazines, and the like -- using XML.

SMIL
Synchronized Multimedia Integration Language

SMIL is a W3C recommendation that covers audio, video, and animations. It also addresses the difficult issue of synchronizing the playback of such elements.

For more information on SMIL, see `http://www.w3.org/TR/REC-smil`.

MathML
Mathematical Markup Language

MathML is a W3C recommendation that deals with the representation of mathematical formulas.

For more information on MathML, see `http://www.w3.org/TR/REC-MathML`.

SVG
Scalable Vector Graphics

SVG is a W3C working draft that covers the representation of vector graphic images. (Vector graphic images that are built from commands that say things like "draw a line (square, circle) from point x,y to point m,n" rather than encoding the image as a series of bits. Such images are more easily scalable, although they typically require more processing time to render.)

For more information on SVG, see `http://www.w3.org/TR/WD-SVG`.

DrawML
Drawing Meta Language

DrawML is a W3C note that covers 2D images for technical illustrations. It also addresses the problem of updating and refining such images.

For more information on DrawML, see http://www.w3.org/TR/NOTE-drawml.

eCommerce Standards

These standards are aimed at using XML in the world of business-to-business (B2B) and business-to-consumer (B2C) commerce.

ICE
Information and Content Exchange

ICE is a protocol for use by content syndicators and their subscribers. It focuses on "automating content exchange and reuse, both in traditional publishing contexts and in business-to-business relationships".

For more information on ICE, see http://www.w3.org/TR/NOTE-ice.

ebXML
Electronic Business with XML

This standard aims at creating a modular electronic business framework using XML. It is the product of a joint initiative by the United Nations (UN/CEFACT) and the Organization for the Advancement of Structured Information Systems (OASIS).

For more information on ebXML, see http://www.ebxml.org/.

cxml
Commerce XML

cxml is a RosettaNet (`www.rosettanet.org`) standard for setting up interactive online catalogs for different buyers, where the pricing and product offerings are company specific. Includes mechanisms to handle purchase orders, change orders, status updates, and shipping notifications.

For more information on cxml, see http://www.cxml.org/

CBL
Common Business Library

CBL is a library of element and attribute definitions maintained by CommerceNet (`www.commerce.net`).

For more information on CBL and a variety of other initiatives that work together to enable eCommerce applications, see `http://www.commerce.net/projects/currentprojects/eco/wg/eCo_Framework_Specifications.html`.

Summary

XML is becoming a widely-adopted standard that is being used in a dizzying variety of application areas. For more information on Java and XML in the open source community, visit http://xml.apache.org/

Top *Contents* *Index* *Glossary*

3. An Overview of the APIs

This page gives you a map so you can find your way around JAXP and the associated XML APIs. The first step is to understand where JAXP fits in with respect to the major Java APIs for XML:

Link Summary

Local Links

- The XML Thread
- Designing an XML Data Structure
- The Simple API for XML (SAX)
- The Document Object Model (DOM)
- Using XSLT
- Examples

API References

- javax.xml.parsers
- org.xml.sax
- org.w3c.dom
- javax.xml.transform

External Links

- http://www.jdom.org
- http://www.dom4j.org
- JDOM JCP Standards Effort: JSR 102

Glossary Terms

DTD, namespace, unparsed entity, URI, URL, URN, W3C

JAXP: Java API for XML Parsing

This API is the subject of the present tutorial. It provides a common interface for creating and using the standard SAX, DOM, and XSLT APIs in Java, regardless of which vendor's implementation is actually being used..

JAXB: Java Architecture for XML Binding

This standard defines a mechanism for writing out Java objects as XML (*marshalling*) and for creating Java objects from such structures (*unmarshalling*). (You compile a class description to create the Java classes, and use those classes in your application.)

JDOM: Java DOM

The standard DOM is a very simple data structure that intermixes text nodes, element nodes, processing instruction nodes, CDATA nodes, entity references, and several other kinds of nodes. That makes it difficult to work with in practice, because you are always sifting through collections of nodes, discarding the ones you don't need into order to process the ones you are interested in. JDOM, on the other hand, creates a tree of *objects* from an XML structure. The resulting tree is much easier to use, and it can be created from an XML structure without a compilation step. For more information on JDOM, visit http://www.jdom.org. For information on the Java Community Process (JCP) standards effort for JDOM, see JSR 102.

DOM4J

Although it is not on the JCP standards track, DOM4J is an open-source, object-oriented alternative to DOM that is in many ways ahead of JDOM in terms of implemented features. As such, it represents an excellent alternative for Java developers who need to manipulate XML-based data. For more information on DOM4J, see http://www.dom4j.org.

JAXM: Java API for XML Messaging

The JAXM API defines a mechanism for exchanging *asynchronous* XML-based messages between applications. ("Asynchronous" means "send it and forget it".)

JAX-RPC: Java API for XML-based Remote Process Communications
The JAX-RPC API defines a mechanism for exchanging *synchronous* XML-based messages between applications. ("Synchronous" means "send a message and wait for the reply".)

JAXR: Java API for XML Registries
The JAXR API provides a mechanism for publishing available services in an external registry, and for consulting the registry to find those services.

The JAXP APIs

Now that you know where JAXP fits into the big picture, the remainder of this page discusses the JAXP APIs .

The main JAXP APIs are defined in the `javax.xml.parsers` package. That package contains two vendor-neutral factory classes: SAXParserFactory and DocumentBuilderFactory that give you a SAXParser and a DocumentBuilder, respectively. The DocumentBuilder, in turn, creates DOM-compliant Document object.

The factory APIs give you the ability to plug in an XML implementation offered by another vendor without changing your source code. The implementation you get depends on the setting of the `javax.xml.parsers.SAXParserFactory` and `javax.xml.parsers.DocumentBuilderFactory` system properties. The default values (unless overridden at runtime) point to the reference implementation.

The remainder of this section shows how the different JAXP APIs work when you write an application.

An Overview of the Packages

As discussed in the previous section, the SAX and DOM APIs are defined by XML-DEV group and by the W3C, respectively. The libraries that define those APIs are:

javax.xml.parsers
The JAXP APIs, which provide a common interface for different vendors' SAX and DOM parsers.
org.w3c.dom
Defines the Document class (a DOM), as well as classes for all of the components of a DOM.
org.xml.sax
Defines the basic SAX APIs.
javax.xml.transform
Defines the XSLT APIs that let you transform XML into other forms.

The "Simple API" for XML (SAX) is the event-driven, serial-access mechanism that does element-by-element processing. The API for this level reads and writes XML to a data repository or the Web. For server-side and high-performance apps, you will want to fully understand this level. But for many applications, a minimal understanding will suffice.

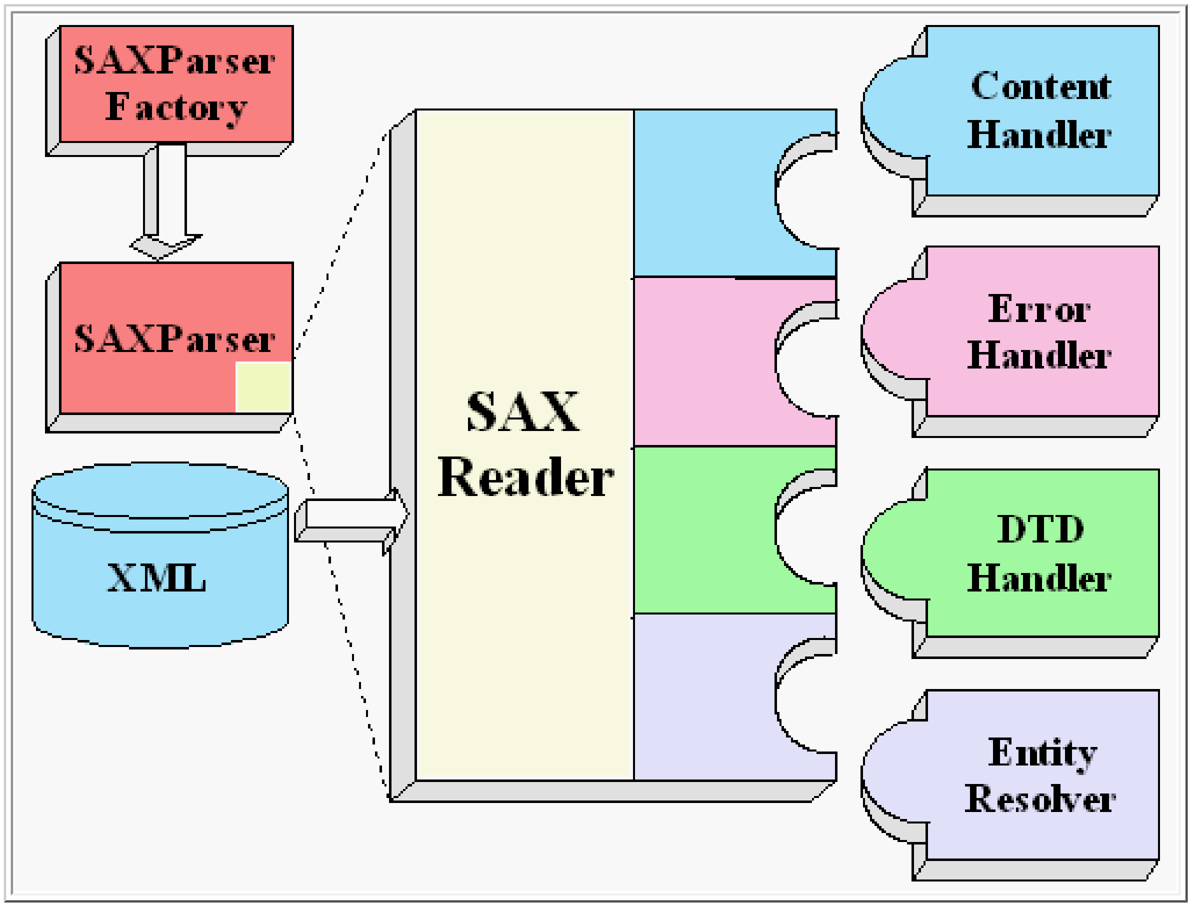
The DOM API is generally an easier API to use. It provides a relatively familiar tree structure of objects. You can use the DOM API to manipulate the hierarchy of application objects it encapsulates. The DOM API is ideal for interactive applications because the entire object model is present in memory, where it can be accessed and manipulated by the user.

On the other hand, constructing the DOM requires reading the entire XML structure and holding the object tree in memory, so it is much more CPU and memory intensive. For that reason, the SAX API will tend to be preferred for server-side applications and data filters that do not require an in-memory representation of the data.

Finally, the XSLT APIs defined in javax.xml.transform let you write XML data to a file or convert it into other forms. And, as you'll see in the XSLT section, of this tutorial, you can even use it in conjunction with the SAX APIs to convert legacy data to XML.

The Simple API for XML (SAX) APIs

The basic outline of the SAX parsing APIs are shown at right. To start the process, an instance of the `SAXParserFactory` classed is used to generate an instance of the parser.



The parser wraps a SAXReader object. When the parser's `parse()` method is invoked, the reader invokes one of several callback methods implemented in the application. Those methods are defined by the interfaces `ContentHandler`, `ErrorHandler`, `DTDHandler`, and `EntityResolver`.

Here is a summary of the key SAX APIs:

`SAXParserFactory`
A SAXParserFactory object creates an instance of the parser determined by the system property, `javax.xml.parsers.SAXParserFactory`.

`SAXParser`
The `SAXParser` interface defines several kinds of `parse()` methods. In general, you pass an XML data source and a DefaultHandler object to the parser, which processes the XML and invokes the appropriate methods in the handler object.

`SAXReader`
The SAXParser wraps a SAXReader. Typically, you don't care about that, but every once in a while you need to get hold of it using SAXParser's `getXMLReader()`, so you can configure it. It is the SAXReader which carries on the conversation with the SAX event handlers you define.

DefaultHandler

Not shown in the diagram, a DefaultHandler implements the `ContentHandler`, `ErrorHandler`, `DTDHandler`, and `EntityResolver` interfaces (with null methods), so you can override only the ones you're interested in.

`ContentHandler`

Methods like `startDocument`, `endDocument`, `startElement`, and `endElement` are invoked when an XML tag is recognized. This interface also defines methods `characters` and `processingInstruction`, which are invoked when the parser encounters the text in an XML element or an inline processing instruction, respectively.

`ErrorHandler`

Methods `error`, `fatalError`, and `warning` are invoked in response to various parsing errors. The default error handler throws an exception for fatal errors and ignores other errors (including validation errors). That's one reason you need to know something about the SAX parser, even if you are using the DOM. Sometimes, the application may be able to recover from a validation error. Other times, it may need to generate an exception. To ensure the correct handling, you'll need to supply your own error handler to the parser.

`DTDHandler`

Defines methods you will generally never be called upon to use. Used when processing a DTD to recognize and act on declarations for an *unparsed entity*.

`EntityResolver`

The `resolveEntity` method is invoked when the parser must identify data identified by a URI. In most cases, a URI is simply a URL, which specifies the location of a document, but in some cases the document may be identified by a URN -- a *public identifier*, or name, that is unique in the web space. The public identifier may be specified in addition to the URL. The `EntityResolver` can then use the public identifier instead of the URL to find the document, for example to access a local copy of the document if one exists.

A typical application implements most of the `ContentHandler` methods, at a minimum. Since the default implementations of the interfaces ignore all inputs except for fatal errors, a robust implementation may want to implement the ErrorHandler methods, as well.

The SAX Packages

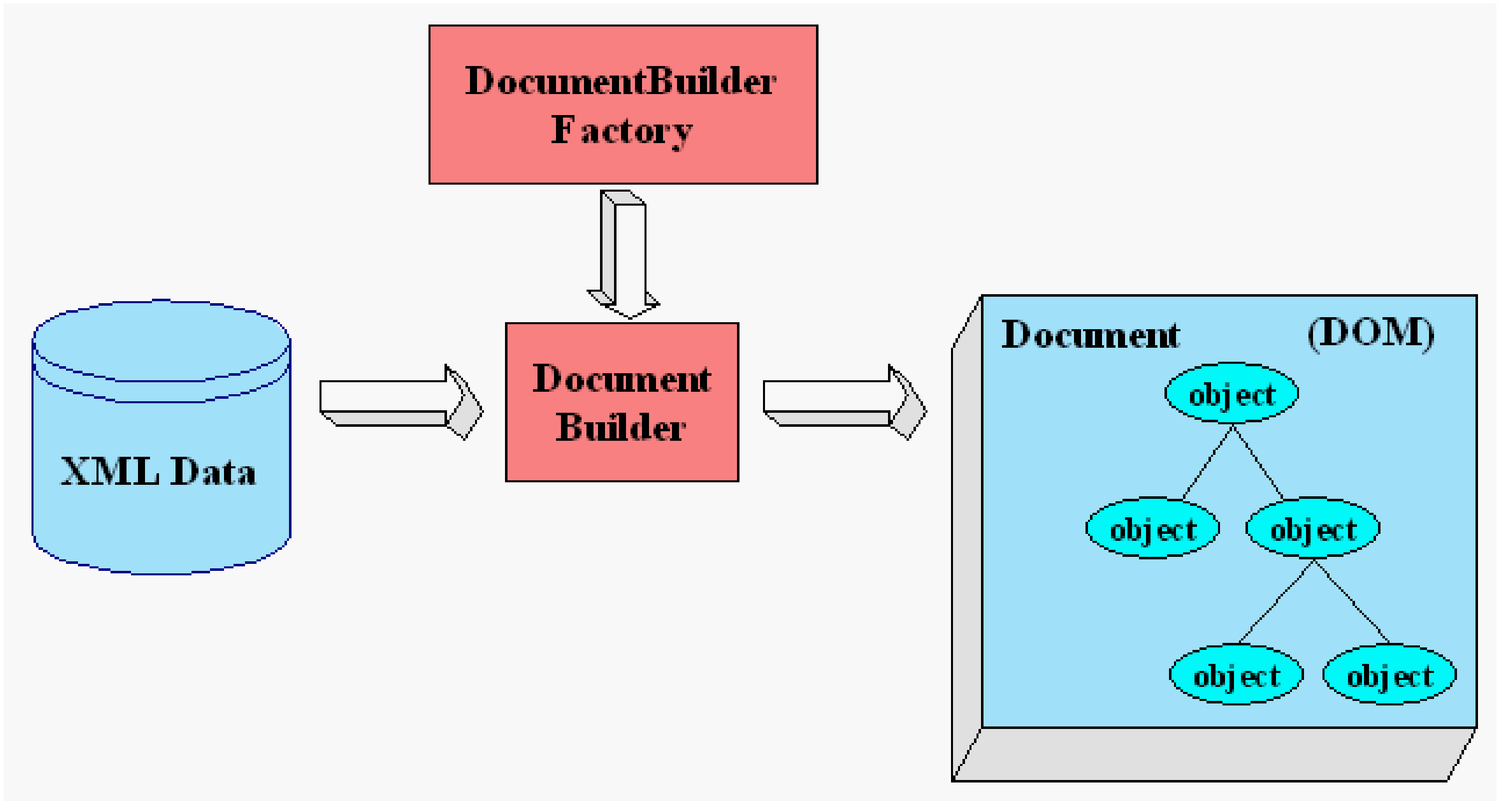
The SAX parser is defined in the following packages.

Package	***Description***
org.xml.sax	Defines the SAX interfaces. The name "`org.xml`" is the package prefix that was settled on by the group that defined the SAX API.
org.xml.sax.ext	Defines SAX extensions that are used when doing more sophisticated SAX processing, for example, to process a document type definitions (DTD) or to see the detailed syntax for a file.

org.xml.sax.helpers	Contains helper classes that make it easier to use SAX -- for example, by defining a default handler that has null-methods for all of the interfaces, so you only need to override the ones you actually want to implement.
javax.xml.parsers	Defines the `SAXParserFactory` class which returns the SAXParser. Also defines exception classes for reporting errors.

The Document Object Model (DOM) APIs

The diagram below shows the JAXP APIs in action:



You use the `javax.xml.parsers.`DocumentBuilderFactory class to get a DocumentBuilder instance, and use that to produce a Document (a DOM) that conforms to the DOM specification. The builder you get, in fact, is determined by the System property, `javax.xml.parsers.DocumentBuilderFactory`, which selects the factory implementation that is used to produce the builder. (The platform's default value can be overridden from the command line.)

You can also use the DocumentBuilder `newDocument()` method to create an empty Document that implements the org.w3c.dom.Document interface. Alternatively, you can use one of the builder's parse methods to create a Document from existing XML data. The result is a DOM tree like that shown in the diagram.

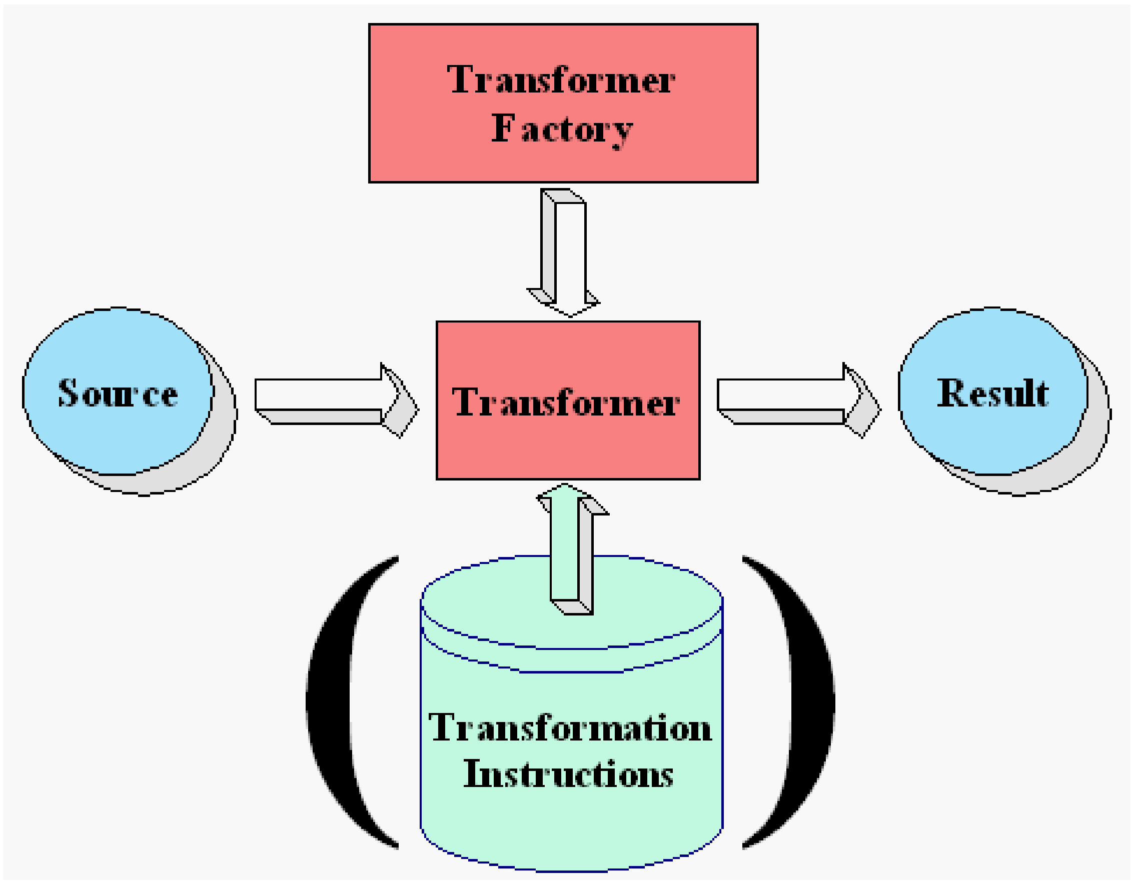
> **Note:**
> Although they are called objects, the entries in the DOM tree are actually fairly low-level data structures. For example, under every *element node* (which corresponds to an XML element) there is a *text node* which contains the name of the element tag! This issue will be explored at length in the DOM section of the tutorial, but users who are expecting objects are usually surprised to find that invoking the `text()` method on an element object returns nothing! For a truly object-oriented tree, see the JDOM API.

The DOM Packages

The Document Object Model implementation is defined in the following packages:

Package	*Description*
org.w3c.dom	Defines the DOM programming interfaces for XML (and, optionally, HTML) documents, as specified by the W3C.
javax.xml.parsers	Defines the DocumentBuilderFactory class and the DocumentBuilder class, which returns an object that implements the W3C Document interface. The factory that is used to create the builder is determined by the `javax.xml.parsers` system property, which can be set from the command line or overridden when invoking the `newInstance` method. This package also defines the `ParserConfigurationException` class for reporting errors.

The XML Style Sheet Translation (XSLT) APIs



The diagram at right shows the XSLT APIs in action.

A `TransformerFactory` object is instantiated, and used to create a `Transformer`. The source object is the input to the transformation process. A source object can be created from SAX reader, from a DOM, or from an input stream.

Similarly, the result object is the result of the transformation process. That object can be a SAX event handler, a DOM, or an output stream.

When the transformer is created, it may be created from a set of transformation instructions, in which case the specified transformations are carried out. If it is created without any specific instructions, then the transformer object simply copies the source to the result.

The XSLT Packages

The XSLT APIs are defined in the following packages:

Package	*Description*

javax.xml.transform	Defines the `TransformerFactory` and `Transformer` classes, which you use to get a object capable of doing transformations. After creating a transformer object, you invoke its `transform()` method, providing it with an input (source) and output (result).
javax.xml.transform.dom	Classes to create input (source) and output (result) objects from a DOM.
javax.xml.transform.sax	Classes to create input (source) from a SAX parser and output (result) objects from a SAX event handler.
javax.xml.transform.stream	Classes to create input (source) and output (result) objects from an I/O stream.

Overview of the JAR Files

Here are the jar files that make up the JAXP bundles, along with the interfaces and classes they contain.

JAR file	*Packages*	*Contents*
jaxp.jar	• javax.xml.parsers • javax.xml.transform ○ javax.xml.transform.dom ○ javax.xml.transform.sax ○ javax.xml.transform.stream	Interfaces
crimson.jar	• org.xml.sax ○ org.xml.sax.helpers ○ org.xml.sax.ext • org.w3c.dom	Interfaces and helper classes
xalan.jar	All of the above	Implementation Classes

Note:
When defining the classpath, specify the jar files in the order shown here: `jaxp.jar`, `crimson.jar`, `xalan.jar`.

Where Do You Go from Here?

At this point, you have enough information to begin picking your own way through the JAXP libraries. Your next step from here depends on what you want to accomplish. You might want to go to:

The XML Thread

If you want to learn more about XML, spending as little time as possible on the Java APIs. (You will see all of the XML sections in the normal course of the tutorial. Follow this thread if you want to bypass the API programming steps.)

Designing an XML Data Structure

If you are creating XML data structures for an application and want some tips on how to proceed. (This is the next step in the XML overview.)

Serial Access with the Simple API for XML (SAX)

If the data structures have already been determined, and you are writing a server application or an XML filter that needs to do the fastest possible processing. This section also takes you step by step through the process of constructing an XML document.

Manipulating Document Contents with the Document Object Model (DOM)

If you need to build an object tree from XML data so you can manipulate it in an application, or convert an in-memory tree of objects to XML. This part of the tutorial ends with a section on namespaces.

Using XSLT

If you need to transform XML tags into some other form, if you want to generate XML output, or if you want to convert legacy data structures to XML.

Browse the Examples

To see some real code. The reference implementation comes with a large number of examples (even though many of them may not make much sense just yet). You can find them in the JAXP `examples` directory, or you can browse to the XML Examples page. The table below divides them into categories depending on whether they are primarily SAX-related, are primarily DOM-related, or serve some special purpose.

Example	***Description***
Sample XML Files	Samples the illustrate how XML files are constructed.
Simple File Parsing	A very short example that creates a DOM using `XmlDocument`'s static `createXmlDocument` method and echoes it to `System.out`. Illustrates the least amount of coding necessary to read in XML data, assuming you can live with all the defaults -- for example, the default error handler, which ignores errors.
Building XML Documents with DOM	A program that creates a Document Object Model in memory and uses it to output an XML structure.
Using SAX	An application that uses the SAX API to echo the content and structure of an XML document using either the validating or non-validating parser, on either a well-formed, valid, or invalid document so you can see the difference in errors that the parsers report. Lets you set the `org.xml.sax.parser` system variable on the command line to determine the parser returned by `org.xml.sax.helpers.ParserFactory`.
XML Namespace Support	An application that reads an XML document into a DOM and echoes its namespaces.
Swing JTree Display	An example that reads XML data into a DOM and populates a JTree.

Text Transcoding	A character set translation example. A document written with one character set is converted to another.

4. Designing an XML Data Structure

This page covers some heuristics you can use when making XML design decisions.

Link Summary

Local Links

- Defining Attributes and Entities in the DTD

External Links

- http://www.XML.org
- http://www.xmlx.com
- http://www.oasis-open.org/cover/elementsAndAttrs.html

Glossary Terms

DTD, entity, external entity, parameter entity

Saving Yourself Some Work

Whenever possible, use an existing DTD. It's usually a lot easier to ignore the things you don't need than to design your own from scratch. In addition, using a standard DTD makes data interchange possible, and may make it possible to use data-aware tools developed by others.

So, if an industry standard exists, consider referencing that DTD with an external parameter entity. One place to look for industry-standard DTDs is at the repository created by the Organization for the Advancement of Structured Information Standards (OASIS) at `http://www.XML.org`. Another place to check is CommerceOne's XML Exchange at `http://www.xmlx.com`, which is described as "a repository for creating and sharing document type definitions".

Note:
Many more good thoughts on the design of XML structures are at the OASIS page, `http://www.oasis-open.org/cover/elementsAndAttrs.html`. If you have any favorite heuristics that can improve this page, please send an email! For the address, see Work in Progress.

Attributes and Elements

One of the issues you will encounter frequently when designing an XML structure is whether to model a given data item as a subelement or as an attribute of an existing element. For example, you could model the title of a slide either as:

```
<slide>
  <title>This is the title</title>
</slide>
```

or as:

```
<slide title="This is the title">...</slide>
```

In some cases, the different characteristics of attributes and elements make it easy to choose. Let's consider those cases first, and then move on to the cases where the choice is more ambiguous.

Forced Choices

Sometimes, the choice between an attribute and an element is forced on you by the nature of attributes and elements. Let's look at a few of those considerations:

The data contains substructures

In this case, the data item must be modeled as an *element*. It can't be modeled as an attribute, because attributes take only simple strings. So if the title can contain emphasized text like this: `The <em>Best</em> Choice`, then the title must be an element.

The data contains multiple lines

Here, it also makes sense to use an *element*. Attributes need to be simple, short strings or else they become unreadable, if not unusable.

The data changes frequently

When the data will be frequently modified, especially by the end user, then it makes sense to model it as an *element*. XML-aware editors tend to make it very easy to find and modify element data. Attributes can be somewhat harder to get to, and therefore somewhat more difficult to modify.

The data is a small, simple string that rarely if ever changes

This is data that can be modeled as an *attribute*. However, just because you *can* does not mean that you should. Check the "Stylistic Choices" section below, to be sure.

The data is confined to a small number of fixed choices

Here is one time when it really makes sense to use an *attribute*. Using the DTD, the attribute can be prevented from taking on any value that is not in the preapproved list. An XML-aware editor can even provide those choices in a drop-down list. Note, though, that the gain in validity restriction comes at a cost in extensibility. The author of the XML document cannot use any value that is not part of the DTD. If another value becomes useful in the future, the DTD will have to be modified before the document author can make use of it.

Stylistic Choices

As often as not, the choices are not as cut and dried as those shown above. When the choice is not forced, you need a sense of "style" to guide your thinking. The question to answer, then, is what makes good XML style, and why.

Defining a sense of style for XML is, unfortunately, as nebulous a business as defining "style" when it comes to art or music. There are a few ways to approach it, however. The goal of this section is to give you some useful thoughts on the subject of "XML style".

Visibility

The first heuristic for thinking about XML elements and attributes uses the concept of *visibility*. If the data is intended to be shown -- to be displayed to some end user -- then it should be modeled as an element. On the other hand, if the information guides XML processing but is never displayed, then it may be better to model it as an attribute. For example, in order-entry data for shoes, shoe size would definitely be an element. On the other hand, a manufacturer's code number would be reasonably modeled as an attribute.

Consumer / Provider

Another way of thinking about the visibility heuristic is to ask who is the consumer and/or provider of the information. The shoe size is entered by a human sales clerk, so it's an element. The manufacturer's code number for a given shoe model, on the other hand, may be wired into the application or stored in a database, so that would be an attribute. (If it were entered by the clerk, though, it should perhaps be an element.) You can also think in terms of who or what is processing the information. Things can get a bit murky at that end of the process, however. If the information "consumers" are order-filling clerks, will they need to see the manufacturer's code number? Or, if an order-filling program is doing all the processing, which data items should be elements in that case? Such philosophical distinctions leave a lot of room for differences in style.

Container vs. Contents

Another way of thinking about elements and attributes is to think of an element as a *container*. To reason by analogy, the *contents* of the container (water or milk) correspond to XML data modeled as elements. On the other hand, *characteristics* of the container (blue or white, pitcher or can) correspond to XML data modeled as attributes. Good XML style will, in some consistent way, separate each container's contents from its characteristics.

To show these heuristics at work: In a slideshow the type of the slide (executive or technical) is best modeled as an attribute. It is a characteristic of the slide that lets it be selected or rejected for a particular audience. The title of the slide, on the other hand, is part of its contents. The visibility heuristic is also satisfied here. When the slide is displayed, the title is shown but the type of the slide isn't. Finally, in this example, the consumer of the title information is the presentation audience, while the consumer of the type information is the presentation program.

Normalizing Data

In the SAX tutorial, the section Defining Attributes and Entities in the DTD shows how to create an external entity that you can reference in an XML document. Such an entity has all the advantages of a modularized routine -- changing that one copy affects every document that references it. The process of eliminating redundancies is known as *normalizing*, so defining entities is one good way to normalize your data.

In an HTML file, the only way to achieve that kind of modularity is with HTML links -- but of course the document is then fragmented, rather than whole. XML entities, on the other hand, suffer no such fragmentation. The entity reference acts like a macro -- the entity's contents are expanded in place, producing a whole document, rather than a fragmented one. And when the entity is defined in an external file, multiple documents can reference it.

The considerations for defining an entity reference, then, are pretty much the same as those you would apply to modularize program code:

1. Whenever you find yourself writing the same thing more than once, think entity. That lets you write it one place and reference it multiple places.
2. If the information is likely to change, especially if it is used in more than one place, definitely think in terms of defining an entity. An example is defining `productName` as an entity so that you can easily change the documents when the product name changes.
3. If the entity will never be referenced anywhere except in the current file, define it in the local_subset of the document's DTD, much as you would define a method or inner class in a program.
4. If the entity will be referenced from multiple documents, define it as an external entity, the same way that would define any generally usable class as an external class.

External entities produce modular XML that is smaller, easier to update and maintain. They can also make the resulting document somewhat more difficult to visualize, much as a good OO design can be easy to change, once you understand it, but harder to wrap your head around at first.

You can also go overboard with entities. At an extreme, you could make an entity reference for the word "the" -- it wouldn't buy you much, but you could do it.

> **Note:**
> The larger an entity is, the less likely it is that changing it will have unintended effects. When you define an external entity that covers a whole section on installation instructions, for example, making changes to the section is unlikely to make any of the documents that depend on it come out wrong. Small inline substitutions can be more problematic, though. For example, if `productName` is defined as an entity, the name change can be to a different part of speech, and that can kill you! Suppose the product name is something like "HtmlEdit". That's a verb. So you write, "You can HtmlEdit your file...". Then, when the official name is decided, it's "Killer". After substitution, that becomes "You can Killer your file...". Argh. Still, even if such simple substitutions can sometimes get you in trouble, they can also save a lot of work. To be totally safe, though, you could set up entities named `productNoun`, `productVerb`, `productAdj`, and `productAdverb`!

Normalizing DTDs

Just as you can normalize your XML document, you can also normalize your DTD declarations by factoring out common pieces and referencing them with a parameter entity. This process is described in the SAX tutorial in Defining Parameter Entities. Factoring out the DTDs (also known as modularizing or normalizing) gives the same advantages and disadvantages as normalized XML -- easier to change, somewhat more difficult to follow.

You can also set up conditionalized DTDs, as described in the SAX tutorial section Conditional Sections. If the number and size of the conditional sections is small relative to the size of the DTD as a whole, that can let you "single source" a DTD that you can use for multiple purposes. If the number of conditional sections gets large, though, the result can be a complex document that is difficult to edit.

XML Alphabetic Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z _

Top ***Contents*** *Index* ***Glossary***

Working with XML: Table of Contents

Top *Contents* ***Index*** ***Glossary***

XML Glossary

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z _

A

archiving
: Saving the state of an object and restoring it.

attribute
: A qualifier on an XML tag that provides additional information. For example, in the tag `<slide title="My Slide">`, `title` is an attribute, and `My Slide` is its value.

B

binary entity
: See unparsed entity.

binding
: Construction of the code needed to process a well-defined bit of XML data.

C

comment
: Text in an XML document that is ignored, unless the parser is specifically told to recognize it. A comment is enclosed in a comment tag, like this: **`<!-- This is a comment -->`**

content
: The part of an XML document that occurs after the prolog, including the root element and everything it contains.

CDATA
: A predefined XML tag for "Character DATA" that says "don't interpret these characters", as

opposed to "Parsed Character Data" (`PCDATA`), in which the normal rules of XML syntax apply (for example, angle brackets demarcate XML tags, tags define XML elements, etc.). CDATA sections are typically used to show examples of XML syntax. Like this:

```
    <![CDATA[ <slide>..A sample slide..</slide> ]]>
```

which displays as:

```
   <slide>..A sample slide.. </slide>
```

D

data

The contents of an element, generally used when the element does not contain any subelements. When it does, the more general term content is generally used. When the only text in an XML structure is contained in simple elements, and elements that have subelements have little or no data mixed in, then that structure is often thought of as XML "data", as opposed to an XML document.

DDP

Document-Driven Programming. The use of XML to define applications.

declaration

The very first thing in an XML document, which declares it as XML. The minimal declaration is `<?xml version="1.0"?>`. The declaration is part of the document prolog.

document

In general, an XML structure in which one or more elements contains text intermixed with subelements. See also: data.

DOM

Document Object Model. A tree of objects with interfaces for traversing the tree and writing an XML version of it, as defined by the W3C specification.

DTD

Document Type Definition. An optional part of the document prolog, as specified by the XML standard. The DTD specifies constraints on the valid tags and tag sequences that can be in the document. The DTD has a number of shortcomings however, which has led to various schema proposals. For example, the DTD entry `<!ELEMENT username (#PCDATA)>` says that the XML element called `username` contains "Parsed Character DATA" -- that is, text alone, with no other structural elements under it. The DTD includes both the local subset, defined in the current file, and the external subset, which consists of the definitions contained in external `.dtd` files that are referenced in the local subset using a parameter entity.

E

element

A unit of XML data, delimited by tags. An XML element can enclose other elements. For example, in the XML structure, "`<slideshow><slide>..</slide><slide>..</slide></slideshow>`", the `<slideshow>` element contains two `<slide>` elements.

entity

A distinct, individual item that can be included in an XML document by referencing it. Such an entity reference can name an entity as small as a character (for example, "`&lt;`", which references the less-than symbol, or left-angle bracket (<). An entity reference can also reference an entire document, or external entity, or a collection of DTD definitions (a parameter entity).

entity reference

A reference to an entity that is substituted for the reference when the XML document is parsed. It may reference a predefined entity like `&lt;` or it may reference one that is defined in the DTD. In the XML data, the reference could be to an entity that is defined in the local subset of the DTD or to an external XML file (an external entity). The DTD can also carve out a segment of DTD specifications and give it a name so that it can be reused (included) at multiple points in the DTD by defining a parameter entity.

error

A SAX parsing error is generally a validation error -- in other words, it occurs when an XML document is not valid, although it can also occur if the declaration specifies an XML version that the parser cannot handle. See also: fatal error, warning.

external entity

An entity that exists as an external XML file, which is included in the XML document using an entity reference.

external subset
That part of the DTD that is defined by references to external `.dtd` files.

F

fatal error
A fatal error occurs in the SAX parser when a document is not well formed, or otherwise cannot be processed. See also: error, warning.

G

general entity
An entity that is referenced as part of an XML document's content, as distinct from a parameter entity, which is referenced in the DTD. A general entity can be a parsed entity or an unparsed entity.

H

HTML
HyperText Markup Language. The language of the Web. A system where every document has a globally unique location, and documents can link to one another.

I

J

K

L

local subset
That part of the DTD that is defined within the current XML file.

M

mixed-content model

A DTD specification that defines an element as containing a mixture of text and one more other elements. The specification must start with `#PCDATA`, followed by alternate elements, and must end with the "zero-or-more" asterisk symbol (*). For example:

```
<!ELEMENT item (#PCDATA | item)* >
```

N

namespace

A standard that lets you specify a unique label to the set of element names defined by a DTD. A document using that DTD can be included in any other document without having a conflict between element names. The elements defined in your DTD are then uniquely identified so that, for example, the parser can tell when an element called `<name>` should be interpreted according to your DTD, rather than using the definition for an element called "name" in a different DTD.

normalization

The process of removing redundancy by modularizing, as with subroutines, and of removing superfluous differences by reducing them to a common denominator. For example, line endings from different systems are normalized by reducing them to a single NL, and multiple whitespace characters are normalized to one space.

notation

A mechanism for defining a data format for a non-XML document referenced as an unparsed entity. This is a holdover from SGML that creaks a bit. The newer standard is to use MIME datatypes and namespaces to prevent naming conflicts.

O

OASIS

Organization for the Advancement of Structured Information Standards. Their home site is http://www.oasis-open.org/. The DTD repository they sponsor is at http://www.XML.org.

P

parameter entity

An entity that consists of DTD specifications, as distinct from a general entity. A parameter entity defined in the DTD can then be referenced at other points, in order to prevent having to recode the definition at each location it is used.

parsed entity

A general entity which contains XML, and which is therefore parsed when inserted into the XML document, as opposed to an unparsed entity.

parser

A module that reads in XML data from an input source and breaks it up into chunks so that your program knows when it is working with a tag, an attribute, or element data. A nonvalidating parser ensures that the XML data is well formed, but does not verify that it is valid.
See also: validating parser.

processing instruction

Information contained in an XML structure that is intended to be interpreted by a specific application.

prolog

The part of an XML document that precedes the XML data. The prolog includes the declaration and an optional DTD.

Q

R

reference

See entity reference

RDF

Resource Description Framework. A standard for defining the kind of data that an XML file contains. Such information could help ensure semantic integrity, for example by helping to make sure that a date is treated as a date, rather than simply as text.

RDF schema

A standard for specifying consistency rules (for example, price must be greater than zero, discount must be less than 15%) that apply to the specifications contained in an RDF.

root

The outermost element in an XML document. The element that contains all other elements.

S

SAX

"Simple API for XML". An event-driven interface in which the parser invokes one of several methods supplied by the caller when a "parsing event" occurs. "Events" include recognizing an XML tag, finding an error, encountering a reference to an external entity, or processing a DTD specification.

schema

A database-inspired method for specifying constraints on XML documents using an XML-based language. Schemas address deficiencies in DTDs, such as the inability to put constraints on the kinds of data that can occur in a particular field (for example, all numeric). Since schemas are founded on XML, they are hierarchical, so it is easier to create an unambiguous specification, and possible to determine the scope over which a comment is meant to apply.

SGML

Standard Generalized Markup Language. The parent of both HTML and XML. However, while HTML shares SGML's propensity for embedding presentation information in the markup, XML is a standard that allows information content to be totally separated from the mechanisms for rendering/displaying that content.

T

tag

A piece of text that describes a unit of data, or element, in XML. The tag is distinguishable as markup, as opposed to data, because it is surrounded by angle brackets (< and >). For example, the element `<name>My Name</name>` has the start tag `<name>`, the end tag `</name>`, which enclose the data "My Name". To treat such markup syntax as data, you use an entity reference or a CDATA section.

U

Unicode

A standard defined by the Unicode Consortium that uses a 16-bit "code page" which maps digits to characters in languages around the world. Because 16 bits covers 32,768 codes, Unicode is large enough to include all the world's languages, with the exception of ideographic languages that have a different character for every concept, like Chinese. For more info, see http://www.unicode.org/.

unparsed entity

A general entity that contains something other than XML. By its nature, then, an unparsed entity contains binary data.

URI

A "Universal Resource Identifier". A URI is either a URL or a URN. (URLs and URNs are concrete entities that actually exist. A "URI" is an abstract superclass -- it's a name we can use when we know we are dealing with either an URL or an URN, and we don't care which.

URL

Universal Resource Locator. A pointer to a specific location (address) on the Web that is unique in all the world. The first part of the URL defines the type of address. For example, `http:/` identifies a Web location. The `ftp:/` prefix identifies a downloadable file. Other prefixes include `file:/` (a file on the local disk system) and `mailto:/` (an email address).

URN

Universal Resource Name. A unique identifier that identifies an entity, but doesn't tell where it is located. That lets the system look it up to see if a local copy exists before going out to find it on the Web. It also allows the web location to change, while still allowing the object to be found.

V

valid

A valid XML document, in addition to being well formed, conforms to all the constraints imposed by a DTD. In other words, it does not contain any tags that are not permitted by the DTD, and the order of the tags conforms to the DTD's specifications.

validating parser

A validating parser is a parser which ensures that an XML document is valid, as well as well-formed.
See also: parser.

W

w3c

The World Wide Web Consortium. The international body that governs Internet standards.

warning

A SAX parser warning is generated when the document's DTD contains duplicate definitions, and similar situations that are not necessarily an error, but which the document author might like to know about, since they could be. See also: fatal error, error.

well-formed

A well-formed XML document is syntactically correct. It does not have any angle brackets that are not part of tags. (The entity references `&lt;` and `&gt;` are used to embed angle brackets in an XML document.) In addition, all tags have an ending tag or are themselves self-ending (`<slide>..</slide>` or `<slide/>`). In addition, in a well-formed document, all tags are fully nested. They never overlap, so this arrangement would produce an error: `<slide><image>..</slide></image>`. Knowing that a document is well formed makes it possible to process it. A well-formed document may not be valid however. To determine that, you need a validating parser and a DTD.

X

XHTML

An XML lookalike for HTML defined by one of several XHTML DTDs. To use XHTML for *everything* would of course defeat the purpose of XML, since the idea of XML is to identify information content, not just tell how to display it. XHTML makes the conversion from HTML to XML, though. You can also reference it in a DTD, which allows you to say, for example, that the text in an element can contain `<em>` and `<b>` tags, rather than being limited to plain text.

XLink

The part of the XLL specification that is concerned with specifying links between documents.

XLL

The XML Link Language specification, consisting of XLink and XPointer.

XML

Extensible Markup Language, which allows you to define the tags (markup) that you need to identify the data and text in XML documents.

XML Schema

The w3c schema specification for XML documents..

XPath
See XSL.

XPointer
The part of the XLL specification that is concerned with identifying sections of documents so that they can referenced in links or included in other documents.

XSL
Extensible Stylesheet Language. An important standard that achieves several goals. XSL lets you:

a. Specify an addressing mechanism, so you can identify the parts of an XML file that a transformation applies to. (**XPath**)
b. Specify tag conversions, so you convert XML data into a different formats. (**XSLT**)
c. Specify display characteristics, such page sizes, margins, and font heights and widths, as well as the *flow objects* on each page. Information fills in one area of a page and then flows automatically flows to the next object when that area fills up. That allows you to wrap text around pictures, for example, or to continue a newsletter article on a different page. (**XML-FO**)

XSL-FO
See XSL.

XSLT
See XSL.

Y

Z

_ (non-alpha)

Part II: Serial Access with the Simple API for XML (SAX)

Link Summary
Local Links
• Manipulating Document Contents with the Document Object Model

In this part of the tutorial, we focus on the event-driven, serial-access mechanism for accessing XML documents, SAX. This is the protocol that most servlets and network-oriented programs will want to use to transmit and receive XML documents, because it's the fastest and least memory-intensive mechanism that is currently available for dealing with XML documents.

On the other hand, the SAX protocol requires a lot more programming than the Document Object Model (DOM). It's also a bit harder to visualize, because it is an event-driven model. (You provide the callback methods, and the parser invokes them as it reads the XML data.) Finally, you can't "back up" to an earlier part of the document, or rearrange it, any more than you can back up a serial data stream or rearrange characters you have read from that stream.

For those reasons, developers who are writing a user-oriented application that displays an XML document and possibly modifies it will want to use the DOM mechanism described in the next part of the tutorial, Manipulating Document Contents with the Document Object Model.

However, even if you plan to do build DOM apps exclusively, there are several important reasons for familiarizing yourself with the SAX model:

- **Same Error Handling**
  When parsing a document for a DOM, the same kinds of exceptions are generated, so the error handling for JAXP SAX and DOM apps are identical.

- **Handling Validation Errors**
  By default, the specifications require that validation errors (which you'll be learning more about in this part of the tutorial) are ignored. If you want to throw an exception in the event of a validation error (and you probably do) then you need to understand how the SAX error handling works.

- **Converting Existing Data**
  As you'll see in the DOM section of the tutorial, Sun's reference implementation provides a mechanism you can use to convert an existing data set to XML -- however, taking advantage of that mechanism requires an understanding the SAX model.

What You'll Learn

This section of the tutorial covers the following topics:

Part III: XML and the Document Object Model (DOM)

Link Summary
Glossary Terms DOM, element, SAX

In the SAX section of the tutorial, you wrote an XML file that contains slides for a presentation. You then used the Simple API for XML (SAX) API to echo the XML to your display.

In this section of the tutorial, you'll use the Document Object Model (DOM) to build a small SlideShow application. You'll start by constructing a DOM and inspecting it, then see how to write a DOM as an XML structure, display it in a GUI, and manipulate the tree structure.

Overview of the Document Object Model

A Document Object Model is a garden-variety tree structure, where each node contains one of the components from an XML structure. The two most common types of nodes are *element nodes* and *text nodes*. Using DOM functions lets you create nodes, remove nodes, change their contents, and traverse the node hierarchy.

What You'll Learn

In this section of the tutorial, you'll parse an existing XML file to construct a DOM, display and inspect the DOM hierarchy, convert the DOM into a user-friendly JTree, and explore the syntax of namespaces. You'll also create a DOM from scratch, and see how to use some of the implementation-specific features in Sun's JAXP reference implementation to convert an existing data set to XML.

This section of the tutorial covers the following topics:

5. Using Namespaces

Part IV: Using XSLT

In this section of the tutorial, you'll learn how to use XSLT to write out a DOM as an XML file. You'll also see how to generate a DOM from an arbitrary data file in order to convert it to XML. Finally, you'll use XSLT to convert XML data into a different form, unlocking the mysteries of the XPath addressing mechanism along the way.

Link Summary
Glossary Terms   XSLT

Overview of the Xml Stylesheet Language for Transformations (XSLT)

XSLT defines mechanisms for addressing XML data (XPath) and for specifying transformations on the data, in order to convert it into other forms

What You'll Learn

In this section of the tutorial, you'll parse an existing XML file to construct a DOM, display and inspect the DOM hierarchy, convert the DOM into a user-friendly JTree, and explore the syntax of namespaces. You'll also create a DOM from scratch, and see how to use some of the implementation-specific features in Sun's JAXP reference implementation to convert an existing data set to XML.

This section of the tutorial covers the following topics:

1. Introducting XSLT and XPath
2. Writing Out a DOM as an XML File
3. Generating XML from an Arbitrary Data Structure
4. Transforming XML Data with XSLT
5. Concatenating XSLT Transformations with a Filter Chain

Top Contents Index Glossary

5a. Creating a Document Type Definition (DTD)

After the XML declaration, the document prolog can include a DTD, which lets you specify the kinds of tags that can be included in your XML document. In addition to telling a validating parser which tags are valid, and in what arrangements, a DTD tells both validating and nonvalidating parsers where text is expected, which lets the parser determine whether the whitespace it sees is significant or ignorable.

Link Summary

Exercise Links

- slideshow1a.dtd
- slideshow1a-dtd.html
- slideSample05.xml
- slideSample05-xml.html

Glossary Terms

declaration, DTD, external subset, local subset, mixed-content model, prolog, root, valid

Basic DTD Definitions

When you were parsing the slide show, for example, you saw that the `characters` method was invoked multiple times before and after comments and slide elements. In those cases, the whitespace consisted of the line endings and indentation surrounding the markup. The goal was to make the XML document readable -- the whitespace was not in any way part of the document contents. To begin learning about DTD definitions, let's start by telling the parser where whitespace is ignorable.

> **Note:** The DTD defined in this section is contained in `slideshow1a.dtd`. (The browsable version is slideshow1a-dtd.html.)

Start by creating a file named `slideshow.dtd`. Enter an XML declaration and a comment to identify the file, as shown below:

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

<!-- DTD for a simple "slide show". -->
```

Next, add the text highlight below to specify that a `slideshow` element contains `slide` elements and nothing else:

```
    <!-- DTD for a simple "slide show". -->

    <!ELEMENT slideshow (slide+)>
```

As you can see, the DTD tag starts with `<!` followed by the tag name (`ELEMENT`). After the tag name comes the name of the element that is being defined (`slideshow`) and, in parentheses, one or more items that indicate the valid contents for that element. In this case, the notation says that a `slideshow` consists of one or more `slide` elements.

Without the plus sign, the definition would be saying that a `slideshow` consists of a single `slide` element. Here are the qualifiers you can add to an element definition:

Qualifier	***Name***	***Meaning***
?	Question Mark	Optional (zero or one)
*	Asterisk	Zero or more
+	Plus Sign	One or more

You can include multiple elements inside the parentheses in a comma separated list, and use a qualifier on each element to indicate how many instances of that element may occur. The comma-separated list tells which elements are valid and the order they can occur in.

You can also nest parentheses to group multiple items. For an example, after defining an `image` element (coming up shortly), you could declare that every `image` element must be paired with a `title` element in a slide by specifying `((image, title)+)`. Here, the plus sign applies to the `image/title` pair to indicate that one or more pairs of the specified items can occur.

Defining Text and Nested Elements

Now that you have told the parser something about where *not* to expect text, let's see how to tell it where text *can* occur. Add the text highlighted below to define the `slide`, `title`, `item`, and `list` elements:

```
    <!ELEMENT slideshow (slide+)>
    <!ELEMENT slide (title, item*)>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT item (#PCDATA | item)* >
```

The first line you added says that a slide consists of a `title` followed by zero or more `item` elements. Nothing new there. The next line says that a title consists entirely of *parsed character data* (`PCDATA`). That's known as "text" in most parts of the country, but in XML-speak it's called "parsed character data".

(That distinguishes it from `CDATA` sections, which contain character data that is not parsed.) The "#" that precedes `PCDATA` indicates that what follows is a special word, rather than an element name.

The last line introduces the vertical bar (`|`), which indicates an *or* condition. In this case, either `PCDATA` or an `item` can occur. The asterisk at the end says that either one can occur zero or more times in succession. The result of this specification is known as a mixed-content model, because any number of `item` elements can be interspersed with the text. Such models must always be defined with `#PCDATA` specified first, some number of alternate items divided by vertical bars (`|`), and an asterisk (`*`) at the end.

Limitations of DTDs

It would be nice if we could specify that an `item` contains either text, or text followed by one or more list items. But that kind of specification turns out to be hard to achieve in a DTD. For example, you might be tempted to define an `item` like this:

```
     <!ELEMENT item (#PCDATA | (#PCDATA, item+)) >
```

That would certainly be accurate, but as soon as the parser sees `#PCDATA` and the vertical bar, it requires the remaining definition to conform to the mixed-content model. This specification doesn't, so you get can error that says: `Illegal mixed content model for 'item'. Found &#x28; ...`, where the hex character 28 is the angle bracket the ends the definition.

Trying to double-define the item element doesn't work, either. A specification like this:

```
     <!ELEMENT item (#PCDATA) >
     <!ELEMENT item (#PCDATA, item+) >
```

produces a "duplicate definition" warning when the validating parser runs. The second definition is, in fact, ignored. So it seems that defining a mixed content model (which allows `item` elements to be interspersed in text) is about as good as we can do.

In addition to the limitations of the mixed content model mentioned above, there is no way to further qualify the kind of text that can occur where `PCDATA` has been specified. Should it contain only numbers? Should be in a date format, or possibly a monetary format? There is no way to say in the context of a DTD.

Finally, note that the DTD offers no sense of hierarchy. The definition for the `title` element applies equally to a `slide` title and to an `item` title. When we expand the DTD to allow HTML-style markup in addition to plain text, it would make sense to restrict the size of an `item` title compared to a `slide` title, for example. But the only way to do that would be to give one of them a different name, such as "`item-title`". The bottom line is that the lack of hierarchy in the DTD forces you to introduce a "hyphenation hierarchy" (or its equivalent) in your namespace. All of these limitations are fundamental motivations behind the development of schema-specification standards.

Special Element Values in the DTD

Rather than specifying a parenthesized list of elements, the element definition could use one of two special values: `ANY` or `EMPTY`. The `ANY` specification says that the element may contain any other defined element, or `PCDATA`. Such a specification is usually used for the root element of a general-purpose XML document such as you might create with a word processor. Textual elements could occur in any order in such a document, so specifying `ANY` makes sense.

The `EMPTY` specification says that the element contains no contents. So the DTD for email messages that let you "flag" the message with `<flag/>` might have a line like this in the DTD:

```
<!ELEMENT flag EMPTY>
```

Referencing the DTD

In this case, the DTD definition is in a separate file from the XML document. That means you have to reference it from the XML document, which makes the DTD file part of the external subset of the full Document Type Definition (DTD) for the XML file. As you'll see later on, you can also include parts of the DTD within the document. Such definitions constitute the local subset of the DTD.

> **Note:** The XML written in this section is contained in `slideSample05.xml`. (The browsable version is slideSample05-xml.html.)

To reference the DTD file you just created, add the line highlighted below to your `slideSample.xml` file:

```
<!--  A SAMPLE set of slides  -->

<!DOCTYPE slideshow SYSTEM "slideshow.dtd">

<slideshow
```

Again, the DTD tag starts with "`<!`". In this case, the tag name, `DOCTYPE`, says that the document is a `slideshow`, which means that the document consists of the `slideshow` element and everything within it:

```
<slideshow>
  ...
```

```
    </slideshow>
```

This tag defines the `slideshow` element as the root element for the document. An XML document must have exactly one root element. This is where that element is specified. In other words, this tag identifies the document *content* as a `slideshow`.

The `DOCTYPE` tag occurs after the XML declaration and before the root element. The `SYSTEM` identifier specifies the location of the DTD file. Since it does not start with a prefix like `http:/` or `file:/`, the path is relative to the location of the XML document. Remember the `setDocumentLocator` method? The parser is using that information to find the DTD file, just as your application would to find a file relative to the XML document. A `PUBLIC` identifier could also be used to specify the DTD file using a unique name -- but the parser would have to be able to resolve it

The `DOCTYPE` specification could also contain DTD definitions within the XML document, rather than referring to an external DTD file. Such definitions would be contained in square brackets, like this:.

```
    <!DOCTYPE slideshow SYSTEM "slideshow1.dtd" [
      ...local subset definitions here...
    ]>
```

You'll take advantage of that facility later on to define some entities that can be used in the document.

Top Contents Index Glossary

5c. Defining Attributes and Entities in the DTD

The DTD you've defined so far is fine for use with the nonvalidating parser. It tells where text is expected and where it isn't, which is all the nonvalidating parser is going to pay attention to. But for use with the validating parser, the DTD needs to specify the valid attributes for the different elements. You'll do that in this section, after which you'll define one internal entity and one external entity that you can reference in your XML file.

Link Summary
Exercise Links   • slideshow1b.dtd   • slideshow1b-dtd.html   • slideSample06.xml   • slideSample06-xml.html   • Echo09-06   • slideSample07.xml   • slideSample07-xml.html   • copyright.xml   • copyright-xml.html   • Echo09-07   ***Glossary Terms***   entity, external entity, notation

Defining Attributes in the DTD

Let's start by defining the attributes for the elements in the slide presentation.

> **Note:**
> The XML written in this section is contained in `slideshow1b.dtd`. (The browsable version is slideshow1b-dtd.html.)

Add the text highlighted below to define the attributes for the `slideshow` element:

```
<!ELEMENT slideshow (slide+)>
<!ATTLIST slideshow
            title    CDATA    #REQUIRED
            date     CDATA    #IMPLIED
            author   CDATA    "unknown"
>
<!ELEMENT slide (title, item*)>
```

The DTD tag `ATTLIST` begins the series of attribute definitions. The name that follows `ATTLIST` specifies the element for which the attributes are being defined. In this case, the element is the `slideshow` element. (Note once again the lack of hierarchy in DTD specifications.)

Each attribute is defined by a series of three space-separated values. Commas and other separators are not allowed, so formatting the definitions as shown above is helpful for readability. The first element in each line is the name of the attribute: `title`, `date`, or `author`, in this case. The second element indicates the type of the data: `CDATA` is character data -- unparsed data, once again, in which a left-angle bracket (<) will never be construed as part of an XML tag. The following table presents the valid choices for the attribute type.

Attribute Type	***Specifies...***
(value1 | value2 | ...)	A list of values separated by vertical bars. (Example below)
`CDATA`	"Unparsed character data". (For normal people, a text string.)
`ID`	A name that no other ID attribute shares.
`IDREF`	A reference to an ID defined elsewhere in the document.
`IDREFS`	A space-separated list containing one or more ID references.
`ENTITY`	The name of an entity defined in the DTD.
`ENTITIES`	A space-separated list of entities.
`NMTOKEN`	A valid XML name composed of letters, numbers, hyphens, underscores, and colons.
`NMTOKENS`	A space-separated list of names.
`NOTATION`	The name of a DTD-specified notation, which describes a non-XML data format, such as those used for image files.*

*This is a rapidly obsolescing specification which will be discussed in greater length towards the end of this section.

When the attribute type consists of a parenthesized list of choices separated by vertical bars, the attribute must use one of the specified values. For an example, add the text highlighted below to the DTD:

```
<!ELEMENT slide (title, item*)>
<!ATTLIST slide
            type   (tech | exec | all) #IMPLIED
>
<!ELEMENT title (#PCDATA)>
<!ELEMENT item (#PCDATA | item)* >
```

This specification says that the `slide` element's `type` attribute must be given as `type="tech"`, type="exec", or type="all". No other values are acceptable. (DTD-aware XML editors can use such specifications to present a pop-up list of choices.)

The last entry in the attribute specification determines the attributes default value, if any, and tells whether or not the attribute is required. The table below shows the possible choices.

Specification	***Specifies...***
#REQUIRED	The attribute value must be specified in the document.
#IMPLIED	The value need not be specified in the document. If it isn't, the application will have a default value it uses.
"defaultValue"	The default value to use, if a value is not specified in the document.
#FIXED "fixedValue"	The value to use. If the document specifies any value at all, it must be the same.

Defining Entities in the DTD

So far, you've seen predefined entities like &amp; and you've seen that an attribute can reference an entity. It's time now for you to learn how to define entities of your own.

> **Note:** The XML defined here is contained in slideSample06.xml. (The browsable version is slideSample06-xml.html.) The output is shown in Echo09-06.

Add the text highlighted below to the DOCTYPE tag in your XML file:

```
<!DOCTYPE slideshow SYSTEM "slideshow1.dtd" [
  <!ENTITY product  "WonderWidget">
  <!ENTITY products "WonderWidgets">
]>
```

The ENTITY tag name says that you are defining an entity. Next comes the name of the entity and its definition. In this case, you are defining an entity named "product" that will take the place of the product name. Later when the product name changes (as it most certainly will), you will only have to change the name one place, and all your slides will reflect the new value.

The last part is the substitution string that replaces the entity name whenever it is referenced in the XML document. The substitution string is defined in quotes, which are not included when the text is inserted into the document.

Just for good measure, we defined two versions, one singular and one plural, so that when the marketing mavens come up with "Wally" for a product name, you will be prepared to enter the plural as "Wallies" and have it substituted correctly.

> **Note:** Truth be told, this is the kind of thing that really belongs in an external DTD. That way, all your documents can reference the new name when it changes. But, hey, this is an example...

Now that you have the entities defined, the next step is to reference them in the slide show. Make the changes highlighted below to do that:

```
<slideshow
    title="WonderWidget&product; Slide Show"
    ...

    <!-- TITLE SLIDE -->
    <slide type="all">
       <title>Wake up to WonderWidgets&products;!</title>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <title>Overview</title>
      <item>Why <em>WonderWidgets&products;</em> are great</item>
      <item/>
      <item>Who <em>buys</em> WonderWidgets&products;</item>
    </slide>
```

The points to notice here are that entities you define are referenced with the same syntax (`&entityName;`) that you use for predefined entities, and that the entity can be referenced in an attribute value as well as in an element's contents.

Echoing the Entity References

When you run the Echo program on this version of the file, here is the kind of thing you see:

```
ELEMENT: <title>
CHARS:   Wake up to
CHARS:   WonderWidgets
CHARS:   !
END_ELM: </title>
```

Note that the existence of the entity reference generates an extra call to the `characters` method, and that the text you see is what results from the substitution.

Additional Useful Entities

Here are three other examples for entity definitions that you might find useful when you write an XML document:

```
<!ENTITY ldquo  "&#147;"> <!-- Left Double Quote -->
<!ENTITY rdquo  "&#148;"> <!-- Right Double Quote -->
<!ENTITY trade  "&#153;"> <!-- Trademark Symbol (TM) -->
<!ENTITY rtrade "&#174;"> <!-- Registered Trademark (R) -->
<!ENTITY copyr  "&#169;"> <!-- Copyright Symbol -->
```

Referencing External Entities

You can also use the `SYSTEM` or `PUBLIC` identifier to name an entity that is defined in an external file. You'll do that now.

> **Note:** The XML defined here is contained in `slideSample07.xml` and in `copyright.xml`. (The browsable versions are slideSample07-xml.html and `copyright-xml.html`.) The Echo output is shown in `Echo09-07`.

To reference an external entity, add the text highlighted below to the `DOCTYPE` statement in your XML file:

```
<!DOCTYPE slideshow SYSTEM "slideshow.dtd" [
  <!ENTITY product  "WonderWidget">
  <!ENTITY products "WonderWidgets">
  <!ENTITY copyright SYSTEM "copyright.xml">
]>
```

This definition references a copyright message contained in a file named `copyright.xml`. Create that file and put some interesting text in it, perhaps something like this:

```
<!--  A SAMPLE copyright  -->
This is the standard copyright message that our lawyers
make us put everywhere so we don't have to shell out a
million bucks every time someone spills hot coffee in their
lap...
```

Finally, add the text highlighted below to your `slideSample.xml` file to reference the external entity:

```
<!-- TITLE SLIDE -->
  ...
</slide>

<!-- COPYRIGHT SLIDE -->
<slide type="all">
   <item>&copyright;</item>
</slide>
```

You could also use an external entity declaration to access a servlet that produces the current date using a definition something like this:

```
 <!ENTITY currentDate SYSTEM
     "http://www.example.com/servlet/CurrentDate?fmt=dd-MMM-yyyy">
```

You would then reference that entity the same as any other entity:

```
Today's date is &currentDate;.
```

Echoing the External Entity

When you run the Echo program on your latest version of the slide presentation, here is what you see:

```
        ...
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <item>
            CHARS:
This is the standard copyright message that our lawyers
make us put everywhere so we don't have to shell out a
million bucks every time someone spills hot coffee in their
lap...
            END_ELM: </item>
        END_ELM: </slide>
        ...
```

Note that the newline which follows the comment in the file is echoed as a character, but that the comment itself is ignored. That is the reason that the copyright message appears to start on the next line after the `CHARS:` label, instead of immediately after the label -- the first character echoed is actually the newline that follows the comment.

Summarizing Entities

An entity that is referenced in the document content, whether internal or external, is termed a general entity. An entity that contains DTD specifications that are referenced from within the DTD is termed a parameter entity. (More on that later.)

An entity which contains XML (text and markup), and which is therefore parsed, is known as a parsed entity. An entity which contains binary data (like images) is known as an unparsed entity. (By its very nature, it must be external.) We'll be discussing references to unparsed entities in the next section of this tutorial.

1. Writing a Simple XML File

Let's start out by writing up a simple version of the kind of XML data you could use for a slide presentation. In this exercise, you'll use your text editor to create the data in order to become comfortable with the basic format of an XML file. You'll be using this file and extending it in later exercises.

Creating the File

Using a standard text editor, create a file called `slideSample.xml`.

> **Note:** Here is a version of it that already exists: slideSample01.xml. (The browsable version is slideSample01-xml.html.) You can use this version to compare your work, or just review it as you read this guide.

Writing the Declaration

Next, write the declaration, which identifies the file as an XML document. The declaration starts with the characters "<?", which is the standard XML identifier for a *processor instruction*. (You'll see other processor instructions later on in this tutorial.)

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

This line identifies the document as an XML document that conforms to version 1.0 of the XML specification, and says that it uses the 8-bit Unicode character-encoding scheme. (For information on encoding schemes, see Java's Encoding Schemes.)

Link Summary

Local Links

- A Quick Introduction to XML
- Java's Encoding Schemes
- The XML Prolog
- Using a LexicalHandler
- Parsing the Parameterized DTD

Exercise Links

- slideSample01.xml
- slideSample01-xml.html

Glossary Terms

attribute, declaration, DTD, element, namespace, tag, XHTML

Since the document has not been specified as "standalone", the parser assumes that it may contain references to other documents. (To see how to specify a document as "standalone", see A Quick Introduction to XML, The XML Prolog.)

Adding a Comment

Comments are ignored by XML parsers. You never see them in fact, unless you activate special settings in the parser. You'll see how to do that later on in the tutorial, when we discuss Using a LexicalHandler. For now, add the text highlighted below to put a comment into the file.

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

<!-- A SAMPLE set of slides -->
```

Defining the Root Element

After the declaration, every XML file defines exactly one element, known as the *root element*. Any other elements in the file are contained *within* that element. Enter the text highlighted below to define the root element for this file, `slideshow`:

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

<!-- A SAMPLE set of slides -->

<slideshow>

</slideshow>
```

Note:
XML element names are case-sensitive. The end-tag must exactly match the start-tag.

Adding Attributes to an Element

A slide presentation has a number of associated data items, none of which require any structure. So it is natural to define them as attributes of the `slideshow` element. Add the text highlighted below to set up some attributes:

```
...
<slideshow
    title="Sample Slide Show"
    date="Date of publication"
```

```
        author="Yours Truly"
        >

    </slideshow>
```

When you create a name for a tag or an attribute, you can use hyphens ("-"), underscores ("_"), colons (":"), and periods (".") in addition to characters and numbers. Unlike HTML, values for XML attributes are always in quotation marks, and multiple attributes are never separated by commas.

> **Note:**
> Colons should be used with care or avoided altogether, because they are used when defining the namespace for an XML document.

Adding Nested Elements

XML allows for hierarchically structured data, which means that an element can contain other elements. Add the text highlighted below to define a slide element and a title element contained within it:

```
    <slideshow
        ...
        >

        <!-- TITLE SLIDE -->
        <slide type="all">
            <title>Wake up to WonderWidgets!</title>
        </slide>

    </slideshow>
```

Here you have also added a *type* attribute to the slide. The idea of this attribute is that slides could be earmarked for a mostly technical or mostly executive audience with `type="tech"` or `type="exec"`, or identified as suitable for both with `type="all"`.

More importantly, though, this example illustrates the difference between things that are more usefully defined as elements (the *title* element) and things that are more suitable as attributes (the *type* attribute). The visibility heuristic is primarily at work here. The title is something the audience will see. So it is an element. The type, on the other hand, is something that never gets presented, so it is an attribute. Another way to think about that distinction is that an element is a container, like a bottle. The type is a characteristic of the *container* (is it tall or short, wide or narrow). The title is a characteristic of the *contents* (water, milk, or tea). These are not hard and fast rules, of course, but they can help when you design your own XML structures.

Adding HTML-Style Text

Since XML lets you define any tags you want, it makes sense to define a set of tags that look like HTML. The XHTML standard does exactly that, in fact. You'll see more about that towards the end of the SAX tutorial. For now, type the text highlighted below to define a slide with a couple of list item entries that use an HTML-style `<em>` tag for emphasis (usually rendered as italicized text):

```
         ...
        <!-- TITLE SLIDE -->
        <slide type="all">
            <title>Wake up to WonderWidgets!</title>
        </slide>


        <!-- OVERVIEW -->
        <slide type="all">
            <title>Overview</title>
            <item>Why <em>WonderWidgets</em> are great</item>
            <item>Who <em>buys</em> WonderWidgets</item>
        </slide>

    </slideshow>
```

We'll see later that defining a *title* element conflicts with the XHTML element that uses the same name. We'll discuss the mechanism that produces the conflict (the DTD) and several possible solutions when we cover Parsing the Parameterized DTD.

Adding an Empty Element

One major difference between HTML and XML, though, is that all XML must be *well-formed* -- which means that every tag must have an ending tag or be an empty tag. You're getting pretty comfortable with ending tags, by now. Add the text highlighted below to define an empty list item element with no contents:

```
         ...
        <!-- OVERVIEW -->
        <slide type="all">
            <title>Overview</title>
            <item>Why <em>WonderWidgets</em> are great</item>
            <item/>
            <item>Who <em>buys</em> WonderWidgets</item>
        </slide>
```

```
</slideshow>
```

Note that any element can be empty element. All it takes is ending the tag with "`/>`" instead of "`>`". You could do the same thing by entering `<item></item>`, which is equivalent.

> **Note:**
> Another factor that makes an XML file *well-formed* is proper nesting. So `<b><i>some text</i></b>` is well-formed, because the `<i>...</i>` sequence is completely nested within the `<b>..</b>` tag. This sequence, on the other hand, is not well-formed: `<b><i>some text</b></i>`.

The Finished Product

Here is the completed version of the XML file:

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

<!--  A SAMPLE set of slides  -->

<slideshow
    title="Sample Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- TITLE SLIDE -->
    <slide type="all">
      <title>Wake up to WonderWidgets!</title>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
        <title>Overview</title>
        <item>Why <em>WonderWidgets</em> are great</item>
        <item/>
        <item>Who <em>buys</em> WonderWidgets</item>
    </slide>
</slideshow>
```

Now that you've created a file to work with, you're ready to write a program to echo it using the SAX parser. You'll do that in the next section.

[Top](#) [Contents](#) [Index](#) [Glossary](#)

4. Substituting and Inserting Text

The next thing we want to with the parser is to customize it a bit, so you can see how to get information it usually ignores. But before we can do that, you're going to need to learn a few more important XML concepts. In this section, you'll learn about:

- Handling Special Characters ("<", "&", and so on)
- Handling Text with XML-style syntax

Link Summary

Exercise Links

- slideSample03.xml
- slideSample03-xml.html
- Echo07-03
- slideSample04.xml
- slideSample04-xml.html
- Echo07-04

API Links

- LexicalHandler

Glossary Terms

CDATA, DTD, entity, entity reference

Handling Special Characters

In XML, an entity is an XML structure (or plain text) that has a name. Referencing the entity by name causes it to be inserted into the document in place of the entity reference. To create an entity reference, the entity name is surrounded by an ampersand and a semicolon, like this:

```
&entityName;
```

Later, when you learn how to write a DTD, you'll see that you can define your own entities, so that `&yourEntityName;` expands to all the text you defined for that entity. For now, though, we'll focus on the predefined entities and character references that don't require any special definitions.

Predefined Entities

An entity reference like `&amp;` contains a name (in this case, “amp”) between the start and end delimiters. The text it refers to (&) is substituted for the name, like a macro in a C or C++ program. The following table shows the predefined entities for special characters.

Character	*Reference*
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&apos;

Character References

A character reference like `&#147;` contains a hash mark (#) followed by a number. The number is the Unicode value for a single character, such as 65 for the letter "A", 147 for the left-curly quote, or 148 for the right-curly quote. In this case, the "name" of the entity is the hash mark followed by the digits that identify the character.

Using an Entity Reference in an XML Document

Suppose you wanted to insert a line like this in your XML document:

> Market Size < predicted

The problem with putting that line into an XML file directly is that when the parser sees the left-angle bracket (<), it starts looking for a tag name, which throws off the parse. To get around that problem, you put `&lt;` in the file, instead of "<".

> **Note:** The results of the modifications below are contained in `slideSample03.xml`. (The browsable version is slideSample03-xml.html.) The results of processing it are shown in `Echo07-03`.

If you are following the programming tutorial, add the text highlighted below to your `slideSample.xml` file:

```
<!-- OVERVIEW -->
<slide type="all">
  <title>Overview</title>
  ...
</slide>

<slide type="exec">
  <title>Financial Forecast</title>
```

```
      <item>Market Size &lt; predicted</item>
      <item>Anticipated Penetration</item>
      <item>Expected Revenues</item>
      <item>Profit Margin </item>
    </slide>

</slideshow>
```

When you run the Echo program on your XML file, you see the following output:

```
ELEMENT: <item>
CHARS:   Market Size
CHARS:   <
CHARS:    predicted
END_ELM: </item>
```

The parser converted the reference into the entity it represents, and passed the entity to the application.

Handling Text with XML-Style Syntax

When you are handling large blocks of XML or HTML that include many of the special characters, it would be inconvenient to replace each of them with the appropriate entity reference. For those situations, you can use a `CDATA` section.

> **Note:** The results of the modifications below are contained in `slideSample04.xml`. (The browsable version is slideSample04-xml.html.) The results of processing it are shown in `Echo07-04`.

A `CDATA` section works like `<pre>...</pre>` in HTML, only more so -- all whitespace in a `CDATA` section is significant, and characters in it are not interpreted as XML. A `CDATA` section starts with `<![CDATA[` and ends with `]]>`. Add the text highlighted below to your `slideSample.XML` file to define a `CDATA` section for a fictitious technical slide:

```
     ...
    <slide type="tech">
      <title>How it Works</title>
      <item>First we fozzle the frobmorten</item>
      <item>Then we framboze the staten</item>
      <item>Finally, we frenzle the fuznaten</item>
      <item><![CDATA[Diagram:

           frobmorten <------------ fuznaten
```

```
                  |             <3>        ^
                  | <1>                    |   <1> = fozzle
                  V                        |   <2> = framboze
                Staten+   <3> = frenzle
                             <2>
         ]]></item>
       </slide>
   </slideshow>
```

When you run the Echo program on the new file, you see the following output:

```
    ELEMENT: <item>
    CHARS:   Diagram:

   frobmorten <------------ fuznaten
      |             <3>        ^
      | <1>                    |   <1> = fozzle
      V                        |   <2> = framboze
    Staten+   <3> = frenzle
                  <2>

   END_ELM: </item>
```

You can see here that the text in the CDATA section arrived as one entirely uninterpreted character string.

Handling CDATA and Other Characters

The existence of CDATA makes the proper echoing of XML a bit tricky. If the text to be output is *not* in a CDATA section, then any angle brackets, ampersands, and other special characters in the text should be replaced with the appropriate entity reference. (Replacing left angle brackets and ampersands is most important, other characters will be interpreted properly without misleading the parser.)

But if the output text *is* in a CDATA section, then the substitutions should not occur, to produce text like that in the example above. In a simple program like our Echo application, it's not a big deal. But any realistic kind of XML-filtering application will want to keep track of whether it is in a CDATA section, in order to treat characters properly.

One other area to watch for is attributes. The text of an attribute value could also contain angle brackets and semicolons that need to be replaced by entity references. (Attribute text can never be in a CDATA section, though, so there is never any question about doing that substitution.)

Later in this tutorial, you will see how to use a LexicalHandler to find out whether or not you are processing a CDATA section. Next, though, you will see how to define a DTD.

8. Handling Lexical Events

You saw earlier that if you are writing text out as XML, you need to know if you are in a `CDATA` section. If you are, then angle brackets (<) and ampersands (&) should be output unchanged. But if you're not in a `CDATA` section, they should be replaced by the predefined entities `&lt;` and `&amp;`. But how do you know if you're processing a `CDATA` section?

Then again, if you are filtering XML in some way, you would want to pass comments along. Normally the parser ignores comments. How can you get comments so that you can echo them?

Finally, there are the parsed entity definitions. If an XML-filtering app sees `&myEntity;` it needs to echo the same string -- not the text that is inserted in its place. How do you go about doing that?

This section of the tutorial answers those questions. It shows you how to use `org.xml.sax.ext.LexicalHandler` to identify comments, `CDATA` sections, and references to parsed entities.

Comments, `CDATA` tags, and references to parsed entities constitute *lexical* information -- that is, information that concerns the text of the XML itself, rather than the XML's information content. Most applications, of course, are concerned only with the *content* of an XML document. Such apps will not use the `LexicalEventListener` API. But apps that output XML text will find it invaluable.

Link Summary
Local Links   • An Overview of the Java XML APIs   ***Exercise Links***   • Echo11.java   • Echo11-09   • Echo12.java   • slideSample10.xml   • slideSample10-xml.html   • Echo12-10   ***API References***   • DeclHandler   • LexicalHandler   ***Glossary Terms***   DOM

> **Note:**
> Lexical event handling is a optional parser feature. Parser implementations are not required to support it. (The reference implementation does so.) This discussion assumes that the parser you are using does so, as well.

How the LexicalHandler Works

To be informed when the SAX parser sees lexical information, you configure the `XmlReader` that underlies the parser with a `LexicalHandler`. (For an overview of those APIs, see An Overview of the Java XML APIs.) The `LexicalHandler` interface defines these even-handling methods:

- `comment(String comment)`
  Passes comments to the application.
- `startCDATA()`, `endCDATA()`
  Tells when a `CDATA` section is starting and ending, which tells your application what kind of characters to expect the next time `characters()` is called.
- `startEntity(String name)`, `endEntity(String name)`
  Gives the name of a parsed entity.
- `startDTD(String name, String publicId, String systemId)`, `endDTD()`
  Tells when a DTD is being processed, and identifies it.

Working with a LexicalHandler

In the remainder of this section, you'll convert the Echo app into a lexical handler and play with its features.

> **Note:**
> The code shown in this section is in `Echo11.java`. The output is shown in `Echo11-09`.

To start, add the code highlighted below to implement the `LexicalHandler` interface and add the appropriate methods.

```
import org.xml.sax.ext.LexicalHandler;

public class Echo extends HandlerBase
   implements LexicalHandler
{
   public static void main(String argv[])
    {
        ...
        // Use an instance of ourselves as the SAX event handler
        ~~DefaultHandler handler = new Echo11();~~
        Echo handler = new Echo();
        ...
```

At this point, the Echo class extends one class and implements an additional interface. You changed the class of the handler variable accordingly, so you can use the same instance as either a DefaultHandler or a LexicalHandler, as appropriate..

Next, add the code highlighted below to get the `XMLReader` that the parser delegates to, and configure it to send lexical events to your lexical handler:

```
    public static void main(String argv[])
    {
        ...
        try {
            ...
            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            XMLReader xmlReader = saxParser.getXMLReader();
            xmlReader.setProperty(
                "http://xml.org/sax/properties/lexical-handler",
                 handler
                 );
            saxParser.parse( new File(argv[0]), handler);
        } catch (SAXParseException spe) {
            ...
```

Here, you configured the XMLReader using the `setProperty()` method defined in the XMLReader class. The property name, defined as part of the SAX standard, is the URL, `http://xml.org/sax/properties/lexical-handler`.

Finally, add the code highlighted below to define the appropriate methods that implement the interface.

```
  public void processingInstruction(String target, String data)
    ...
  }

  public void comment(char[] ch, int start, int length)
  throws SAXException
  {
  }

  public void startCDATA()
  throws SAXException
  {
  }

  public void endCDATA()
  throws SAXException
  {
  }

  public void startEntity(String name)
  throws SAXException
  {
  }

  public void endEntity(String name)
  throws SAXException
  {
  }

  public void startDTD(String name, String publicId, String systemId)
  throws SAXException
  {
  }

  public void endDTD()
  throws SAXException
  {
  }

  private void emit(String s)
    ...
```

You have now turned the `Echo` class into a lexical handler. In the next section, you'll start experimenting with lexical events.

Echoing Comments

The next step is to do something with one of the new methods. Add the code highlighted below to echo comments in the XML file:

```
      public void comment(String text)
            throws SAXException
      {
        String text = new String(ch, start, length);
        nl(); emit("COMMENT: "+text);
      }
```

When you compile the Echo program and run it on your XML file, the result looks something like this:

```
     COMMENT:   A SAMPLE set of slides
     COMMENT:  FOR WALLY / WALLIES
     COMMENT:
         DTD for a simple "slide show".

     COMMENT:  Defines the %inline; declaration
     COMMENT:  ...
```

The line endings in the comments are passed as part of the comment string, once again normalized to newlines (`\n`). You can also see that comments in the DTD are echoed along with comments from the file. (That can pose problems when you want to echo only comments that are in the data file. To get around that problem, you can use the `startDTD` and `endDTD` methods.)

Echoing Other Lexical Information

To finish up this section, you'll exercise the remaining `LexicalHandler` methods.

> **Note:**
> The code shown in this section is in `Echo12.java`. The file it operates on is `slideSample10.xml`. (The browsable version is slideSample10-xml.html.) The results of processing are in `Echo12-10`.

Make the changes highlighted below to remove the comment echo (you don't need that any more) and echo the other events:

```
    public void comment(String text)
    throws SAXException
    {
      ~~String text = new String(ch, start, length);~~
      ~~nl(); emit("COMMENT: "+text);~~
    }

    public void startCDATA()
    throws SAXException
    {
      nl(); emit("START CDATA SECTION");
    }

    public void endCDATA()
    throws SAXException
    {
      nl(); emit("END CDATA SECTION");
    }

    public void startEntity(String name)
    throws SAXException
    {
      nl(); emit("START ENTITY: "+name);
    }

    public void endEntity(String name)
    throws SAXException
    {
      nl(); emit("END ENTITY: "+name);
    }

    public void startDTD(String name, String publicId, String systemId)
    throws SAXException
    {
      nl(); emit("START DTD: "+name
              +"\n           publicId=" + publicId
              +"\n           systemId=" + systemId);
    }

    public void endDTD()
    throws SAXException
    {
      nl(); emit("END DTD");
    }
```

Here is what you see when the DTD is processed:

```
START DTD: slideshow
           publicId=null
           systemId=file:/..../samples/slideshow3.dtd
END DTD
```

Note:
To see events that occur while the DTD is being processed, use org.xml.sax.ext.DeclHandler.

Here is what happens when the internally defined `products` entity is processed with the latest version of the program:

```
ELEMENT: <slide-title>
CHARS:   Wake up to
START PARSED ENTITY: products
CHARS:   WonderWidgets
END PARSED ENTITY: products, INCLUDED=true
CHARS:   !
END_ELM: </slide-title>
```

And here is the result of processing the external `copyright` entity:

```
            START PARSED ENTITY: copyright
            CHARS:
This is the standard copyright message ...
            END PARSED ENTITY: copyright, INCLUDED=true
```

Finally, you get output like this for the `CDATA` section:

```
START CDATA SECTION
CHARS:   Diagram:

frobmorten <------------ fuznaten
   |            <3>        ^
   | <1>                   |   <1> = fozzle
   V                       |   <2> = framboze
staten --------------------+   <3> = frenzle
            <2>

END CDATA SECTION
```

In summary, the `LexicalHandler` gives you the event-notifications you need to produce an accurate reflection of the original XML text.

Top Contents Index Glossary

5b. DTD's Effect on the Nonvalidating Parser

In the last section, you defined a rudimentary document type and used it in your XML file. In this section, you'll use the Echo program to see how the data appears to the SAX parser when the DTD is included.

Note:
The output shown in this section is contained in `Echo07-05`.

Link Summary
Exercise Links   • Echo07-05   • Echo08.java   • Echo08-05   • Echo09.java

Running the Echo program on your latest version of `slideSample.xml` shows that many of the superfluous calls to the `characters` method have now disappeared:

```
ELEMENT: <slideshow
   ATTR: ...
>
PROCESS: ...
    ELEMENT: <slide
       ATTR: ...
    >
        ELEMENT: <title>
        CHARS:   Wake up to ...
        END_ELM: </title>
    END_ELM: </slide>
    ELEMENT: <slide
        ATTR: ...
    >
    ...
```

It is evident here that the whitespace characters which were formerly being echoed around the `slide` elements are no longer appearing, because the DTD declares that `slideshow` consists solely of `slide` elements:

```
<!ELEMENT slideshow (slide+)>
```

Tracking Ignorable Whitespace

Now that the DTD is present, the parser is no longer the `characters` method with whitespace that it knows to be irrelevant. From the standpoint of an application that is only interested in processing the XML data, that is great. The application is never bothered with whitespace that exists purely to make the XML file readable.

On the other hand, if you were writing an application that was filtering an XML data file, and you wanted to output an equally readable version of the file, then that whitespace would no longer be irrelevant -- it would be essential. To get those characters, you need to add the `ignorableWhitespace` method to your application. You'll do that next.

> **Note:**
> The code written in this section is contained in `Echo08.java`. The output is in `Echo08-05`.

To process the (generally) ignorable whitespace that the parser is seeing, add the code highlighted below to implement the `ignorableWhitespace` event handler in your version of the Echo program:

```
public void characters (char buf[], int offset, int len)
  ...
}


public void ignorableWhitespace(char buf[], int offset, int Len)
throws SAXException
{
    nl(); emit("IGNORABLE");
}

public void processingInstruction(String target, String data)
```

This code simply generates a message to let you know that ignorable whitespace was seen.

> **Note:**
> Again, not all parsers are created equal. The SAX specification does not require this method to be invoked. The Java XML implementation does so whenever the DTD makes it possible.

When you run the Echo application now, your output looks like this:

```
ELEMENT: <slideshow
   ATTR: ...
>
IGNORABLE
IGNORABLE
PROCESS: ...
IGNORABLE
IGNORABLE
    ELEMENT: <slide
       ATTR: ...
    >
    IGNORABLE
        ELEMENT: <title>
        CHARS:   Wake up to ...
        END_ELM: </title>
    IGNORABLE
    END_ELM: </slide>
IGNORABLE
IGNORABLE
    ELEMENT: <slide
       ATTR: ...
    >
    ...
```

Here, it is apparent that the `ignorableWhitespace` is being invoked before and after comments and slide elements, where `characters` was being invoked before there was a DTD.

Cleanup

Now that you have seen ignorable whitespace echoed, remove that code from your version of the Echo program -- you won't be needing it any more in the exercises ahead.

> **Note:**
> That change has been made in `Echo09.java`.

Documents and Data

Earlier, you learned that one reason you hear about XML *documents*, on the one hand, and XML *data*, on the other, is that XML handles both comfortably, depending on whether text is or is not allowed between elements in the structure.

In the sample file you have been working with, the `slideshow` element is an example of a *data element* -- it contains only subelements with no intervening text. The `item` element, on the other hand, might be termed a *document element*, because it is defined to include both text and subelements.

As you work through this tutorial, you will see how to expand the definition of the title element to include HTML-style markup, which will turn it into a document element as well.

Empty Elements, Revisited

Now that you understand how certain instances of whitespace can be ignorable, it is time revise the definition of an "empty" element. That definition can now be expanded to include

```
<foo>   </foo>
```

where there is whitespace between the tags and the DTD defines that whitespace as ignorable.

Top *Contents* *Index* *Glossary*

6. Using the Validating Parser

By now, you have done a lot of experimenting with the nonvalidating parser. It's time to have a look at the validating parser and find out what happens when you use it to parse the sample presentation.

Two things to understand about the validating parser at the outset are:

a. The DTD is required.
2. Since the DTD is present, the `ignorableWhitespace` method is invoked whenever the DTD makes that possible.

Link Summary
Exercise Links   • Echo10.java   • Echo10-01   • Echo10-06   • Echo10-07   ***API References***   ○ SAXParserFactory   ***Glossary Terms***   DTD, error, XHTML

Configuring the Factory

The first step is modify the Echo program so that it uses the validating parser instead of the nonvalidating parser.

> **Note:**
> The code in this section is contained in `Echo10.java`.

To use the validating parser, make the changes highlighted below:

```
public static void main(String argv[])
{
    if (argv.length != 1) {
        ...
    }
    ~~// Use the default (non-validating) parser~~
    // Use the validating parser
    SAXParserFactory factory = SAXParserFactory.newInstance();
    factory.setValidating(true);
    try {
        ...
```

Here, you configured the factory so that it will produce a validating parser when `newSAXParser` is invoked. You can also configure it to return a namespace-aware parser using `setNamespaceAware(true)`. The reference implementation supports any combination of configuration options. If the combination of .

Changing the Environment Variable

If no other factory class is specified, the default `SAXParserFactory` class is used. To use a different manufacturer's parser, you can change the value of the environment variable that points to it. You can do that from the command line, like this:

```
> java -Djavax.xml.parsers.SAXParserFactory=yourFactoryHere ...
```

The factory name you specify must be a fully qualified class name (all package prefixes included). For more information, see the documentation in the `newInstance()` method of the SAXParserFactory class.

Experimenting with Validation Errors

To see what happens when the XML document does not specify a DTD, remove the `DOCTYPE` statement from the XML file and run the Echo program on it.

> **Note:**
> The output shown here is contained in `Echo10-01`.

The result you see looks like this:

```
<?xml version='1.0' encoding='UTF-8'?>
** Warning, line 5, uri file: ...
   Valid documents must have a <!DOCTYPE declaration.
** Parsing error, line 5, uri file: ...
   Element type "slideshow" is not declared.
```

So now you know that a DTD is a requirement for a valid document. That makes sense. (Note, though, that the lack of a type declaration only generates a warning, as specified in the standard. On the other hand, any attempt to actually parse the document is immediately greeted with an error! Oh well...)

So what happens when you run the parser on your current version of the slide presentation, with the DTD specified?

> **Note:**
> The output shown here is contained in `Echo10-07`.

This time, the parser gives the following error message:

```
** Parsing error, line 28, uri file:...
   Element "slide" does not allow "item" here.
```

This error occurs because the definition of the `slide` element requires a `title`. That element is not optional, and the copyright slide does not have one. To fix the problem, add the question mark highlighted below to make `title` an optional element:

```
<!ELEMENT slide (image?, title?, item*)>
```

Now what happens when you run the program?

> **Note:**
> You could also remove the copyright slide, which produces the same result shown below, as reflected in Echo10-06.

The answer is that everything runs fine, until the parser runs into the `<em>` tag contained in the overview slide. Since that tag was not defined in the DTD, the attempt to validate the document fails. The output looks like this:

```
...
    ELEMENT: <title>
    CHARS:   Overview
    END_ELM: </title>
    ELEMENT: <item>
    CHARS:   Why ** Parsing error, line 24, uri file:...
Element "item" does not allow "em" -- (#PCDATA|item)
org.xml.sax.SAXParseException: Element "item" does not allow "em" -- (#PCDATA|item)
       at com.sun.xml.parser.Parser.error(Parser.java:2798)
...
```

The error message identifies the part of the DTD that caused validation to fail. In this case it is the line that defines an `item` element as `(#PCDATA | item)`.

> **Exercise:** Make a copy of the file and remove all occurrences of `<em>` from it. Can the file be validated now? (In the next section, you'll learn how to define parameter entries so that we can use XHTML in the elements we are defining as part of the slide presentation.)

Error Handling in the Validating Parser

It is important to recognize that the only reason an exception is thrown when the file fails validation is as a result of the error-handling code you entered in the early stages of this tutorial. That code is reproduced below:

```
public void error(SAXParseException e)
throws SAXParseException
{
    throw e;
}
```

If that exception is not thrown, the validation errors are simply ignored.

> **Exercise:** Try commenting out the line that throws the exception. What happens when you run the parser now?

In general, a SAX parsing *error* is a validation error, although we have seen that it can also be generated if the file specifies a version of XML that the parser is not prepared to handle. The thing to remember is that your application will not generate a validation exception unless you supply an error handler like the one above.

2a. Echoing an XML File with the SAX Parser

In real life, you are going to have little need to echo an XML file with a SAX parser. Usually, you'll want to process the data in some way in order to do something useful with it. (If you want to echo it, it's easier to build a DOM tree and use that for output.) But echoing an XML structure is a great way to see the SAX parser in action, and it can be useful for debugging.

In this exercise, you'll echo SAX parser events to `System.out`. Consider it the "Hello World" version of an XML-processing program. It shows you how to use the SAX parser to get at the data, and then echoes it to show you what you've got.

> **Note:**
> The code discussed in this section is in `Echo01.java`. The file it operates on is `slideSample01.xml`. (The browsable version is slideSample01-xml.html.)

Creating the Skeleton

Start by creating a file named `Echo.java` and enter the skeleton for the application:

```
public class Echo
{
    public static void main(String argv[])

    {

    }

}
```

Since we're going to run it standalone, we need a main method. And we need command-line arguments so we can tell the app which file to echo.

Importing Classes

Next, add the import statements for the classes the app will use:

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo
{
  ...
```

Link Summary

Local Links

- An Overview of the Java XML APIs
- Java's Encoding Schemes
- Handling Errors with the Non-Validating Parser
- Substituting and Inserting Text

Examples

- compile, run
- compile.bat, run.bat
- Echo01.java
- slideSample01.xml
- slideSample01-xml.html
- Echo01-01
- Echo02.java
- Echo02-01
- Echo03.java
- Echo03-01

API References

- DefaultHandler
- org.xml.sax
- ContentHandler
- LexicalHandler
- SAXException
- AttributeList

Glossary Terms

DOM

The classes in `java.io`, of course, are needed to do output. The `org.xml.sax` package defines all the interfaces we use for the SAX parser. The SAXParserFactory class creates the instance we use. It throws a ParserConfigurationException if it is unable to produce a parser that matches the specified configuration of options. (You'll see more about the configuration options later.) The SAXParser is what the factory returns for parsing, and the DefaultHandler defines the class that will handle the SAX events that the parser generates.

Setting up for I/O

The first order of business is to process the command line argument, get the name of the file to echo, and set up the output stream. Add the text highlighted below to take care of those tasks and do a bit of additional housekeeping:

```
public static void main(String argv[])

{
    if (argv.length != 1) {
        System.err.println("Usage: cmd filename");
        System.exit(1);
    }
    try {
        // Set up output stream
        out = new OutputStreamWriter(System.out, "UTF8");

    } catch (Throwable t) {
        t.printStackTrace();
    }
    System.exit(0);
}

static private Writer out;
```

When we create the output stream writer, we are selecting the UTF-8 character encoding. We could also have chosen US-ASCII, or UTF-16, which the Java platform also supports. For more information on these character sets, see Java's Encoding Schemes.

Implementing the ContentHandler Interface

The most important interface for our current purposes is the `ContentHandler` interface. That interface requires a number of methods that the SAX parser invokes in response to different parsing events. The major event handling methods are: `startDocument`, `endDocument`, `startElement`, `endElement`, and `characters`.

The easiest way to implement that interface is to extend the DefaultHandler class, defined in the org.xml.sax.helpers package. That class provides do-nothing methods for all of the ContentHandler events . Enter the code highlighted below to extend that class:

```
public class Echo extends DefaultHandler
{
   ...
}
```

Note:
`DefaultHandler` also defines do-nothing methods for the other major events, defined in the `DTDHandler`, `EntityResolver`, and `ErrorHandler` interfaces. You'll learn more about those methods as we go along.

Each of these methods is required by the interface to throw a `SAXException`. An exception thrown here is sent back to the parser, which sends it on to the code that invoked the parser. In the current program, that means it winds up back at the `Throwable` exception handler at the bottom of the `main` method.

When a start tag or end tag is encountered, the name of the tag is passed as a String to the `startElement` or `endElement` method, as appropriate. When a start tag is encountered, any attributes it defines are also passed in an `Attributes` list. Characters found within the element are passed as an array of characters, along with the number of characters (`length`) and an offset into the array that points to the first character.

Setting up the Parser

Now (at last) you're ready to set up the parser. Add the text highlighted below to set it up and get it started:

```
    public static void main(String argv[])

    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        DefaultHandler handler = new Echo();

        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler );

        } catch (Throwable t) {
            t.printStackTrace();
        }
        System.exit(0);
    }
```

With these lines of code, you created a `SAXParserFactory` instance, as determined by the setting of the `javax.xml.parsers.SAXParserFactory` system property. You then got a parser from the factory and gave the parser an instance of this class to handle the parsing events, telling it which input file to process.

> **Note:**
> The `javax.xml.parsers.`SAXParser class is a wrapper that defines a number of convenience methods. It wraps the (somewhat-less friendly) `org.xml.sax.`Parser object. If needed, you can obtain that parser using the SAXParser's `getParser()` method.

For now, you are simply catching any exception that the parser might throw. You'll learn more about error processing in a later section of the tutorial, Handling Errors with the Nonvalidating Parser.

Writing the Output

The `ContentHandler` methods throw `SAXException`s but not `IOException`s, which can occur while writing. The `SAXException` can wrap another exception, though, so it makes sense to do the output in a method that takes care of the exception-handling details. Add the code highlighted below to define an `emit` method that does that:

```
    static private Writer out;

    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
    ...
```

When emit is called, any I/O error is wrapped in SAXException along with a message that identifies it. That exception is then thrown back to the SAX parser. You'll learn more about SAX exceptions later on. For now, keep in mind that emit is a small method that handles the string output. (You'll see it called a lot in the code ahead.)

Spacing the Output

There is one last bit of infrastructure we need before doing some real processing. Add the code highlighted below to define a nl() method that writes the kind of line-ending character used by the current system:

```
private void emit(String s)
  ...

}


private void nl()
throws SAXException
{
    String lineEnd =  System.getProperty("line.separator");
    try {
        out.write(lineEnd);

    } catch (IOException e) {
        throw new SAXException("I/O error", e);
    }
}
```

> **Note:** Although it seems like a bit of a nuisance, you will be invoking nl() many times in the code ahead. Defining it now will simplify the code later on. It also provides a place to indent the output when we get to that section of the tutorial.

Handling Content Events

Finally, let's write some code that actually processes the ContentHandler events. Add the code highlighted below to handle the start-document and end-document events:

```
static private Writer out;


public void startDocument()
throws SAXException
{
    emit("<?xml version='1.0' encoding='UTF-8'?>");
    nl();
}

public void endDocument()
throws SAXException
{
    try {
        nl();
        out.flush();
    } catch (IOException e) {
        throw new SAXException("I/O error", e);
    }
}

private void emit(String s)
...
```

Here, you are echoing an XML declaration when the parser encounters the start of the document. Since you set up the `OutputStreamWriter` using the UTF-8 encoding, you include that specification as part of the declaration.

> **Note:** However, the IO classes don't understand the hyphenated encoding names, so you specified "UTF8" rather than "UTF-8".

At the end of the document, you simply put out a final newline and flush the output stream. Not much going on there. Now for the interesting stuff. Add the code highlighted below to process the start-element and end-element events:

```
    public void startElement(String namespaceURI,
                             String sName, // simple name (localName)
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        String eName = sName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                emit(" ");
                emit(aName+"=\""+attrs.getValue(i)+"\"");
            }
        }
        emit(">");
    }

    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          )
    throws SAXException
    {
        emit("</"+sName+">");
    }

    private void emit(String s)
    ...
```

With this code, you echoed the element tags, including any attributes defined in the start tag. Note that when the startElement() method is invoked, the simple name ("local name") for elements and attributes could turn out to be the empty string, if namespace processing was not enabled. The code handles that case by using the qualifed name whenever the simple name is the empty string.

To finish this version of the program, add the code highlighted below to echo the characters the parser sees:

```
public void characters(char buf[], int offset, int len)
throws SAXException
{
    String s = new String(buf, offset, len);
    emit(s);
}

private void emit(String s)
...
```

Congratulations! You've just written a SAX parser application. The next step is to compile and run it.

> **Note:** To be strictly accurate, the character handler should scan the buffer for ampersand characters ('&') and left-angle bracket characters ('<') and replace them with the strings "`&amp;`" or "`&lt;`", as appropriate. You'll find out more about that kind of processing when we discuss entity references in Substituting and Inserting Text.

Compiling the Program

To compile the program you created, you'll execute the appropriate command for your system (or use one of the command scripts mentioned below):

Windows:

```
javac -classpath %JAXP%\jaxp.jar;%JAXP%\crimson.jar;%JAXP%\xalan.jar Echo.java
```

Unix:

```
javac -classpath ${JAXP}/jaxp.jar:${JAXP}/crimson.jar:${JAXP}/xalan.jar Echo.java
```

where:

- javac is a version 1.2 or later java platform compiler
- `JAXP` is where you installed the JAXP libraries.
- `jaxp.jar` contains the JAXP-specific APIs
- `crimson.jar` contains the interfaces and classes that make up the SAX and DOM APIs, as well as the reference implementation for the parser. (To use a different parser, substitute it here. For example, specify `xerces.jar` to use the parser from apache.org.)
- `xalan.jar` contains the implementation classes for the XSLT `transform` package. (Similarly, substitute this specification to use a different XSLT package.)

Note:
Although Xalan is not strictly needed at this point in the tutorial, you'll be using it later on.

Running the Program

To run the program, you'll once again execute the appropriate command for your system (or use one of the command scripts mentioned below):

Windows:

```
Java -classpath .;%JAXP%\jaxp.jar;%JAXP%\crimson.jar;%JAXP%\xalan.jar Echo
slideSample.xml
```

UNIX:

```
Java -classpath .:${JAXP}/jaxp.jar:${JAXP}/crimson.jar:${JAXP}/xalan.jar Echo
slideSample.xml
```

Command Scripts

To make life easier, here are some command scripts you can use to compile and run your apps as you work through this tutorial.

	UNIX	***Windows***
Scripts	`compile`, `run`	`compile.bat`, `run.bat`
Netscape	Click, choose **File-->Save As**	Right click, choose **Save Link As**.
Internet Explorer	-/-	Right click, choose **Save Target As**.

Checking the Output

The program's output as shown in `Echo01-01`. Here is part of it, showing some of its weird-looking spacing:

```
...
<slideshow title="Sample Slide Show" date="Date of publication" author="Yours Truly">


    <slide type="all">
      <title>Wake up to WonderWidgets!</title>
    </slide>
    ...
```

Looking at this output, a number of questions arise. Namely, where is the excess vertical whitespace coming from? And why is it that the elements are indented properly, when the code isn't doing it? We'll answer those questions in a moment. First, though, there are a few points to note about the output:

- The comment defined at the top of the file

  ```
  <!-- A SAMPLE set of slides -->
  ```

  does not appear in the listing. Comments are ignored by definition, unless you implement a `LexicalHandler`. You'll see more about that later on in this tutorial.

- Element attributes are listed all together on a single line. If your window isn't really wide, you won't see them all.

- The single-tag empty element you defined (`<item/>`) is treated exactly the same as a two-tag empty element (`<item></item>`). It is, for all intents and purposes, identical. (It's just easier to type and consumes less space.)

Identifying the Events

This version of the echo program might be useful for displaying an XML file, but it's not telling you much about what's going on in the parser. The next step is to modify the program so that you see where the spaces and vertical lines are coming from.

> **Note:** The code discussed in this section is in `Echo02.java`. The output it produces is shown in `Echo02-01`.

Make the changes highlighted below to identify the events as they occur:

```
public void startDocument()
throws SAXException
{
    nl();
    nl();
    emit("START DOCUMENT");
    nl();
    emit("<?xml version='1.0' encoding='UTF-8'?>");
    nl();
}

public void endDocument()
throws SAXException
{
    nl(); emit("END DOCUMENT");
    try {
     ...
}

public void startElement(...)
throws SAXException
{
    nl(); emit("ELEMENT: ");
    emit("<"+name);
    if (attrs != null) {
        for (int i = 0; i < attrs.getLength(); i++) {
            emit(" ");
            emit(attrs.getName(i)+"=\""+attrs.getValue(i)+"\"");
            nl();
            emit("   ATTR: ");
            emit(attrs.getLocalName(i));
            emit("\t\"");
            emit(attrs.getValue(i));
            emit("\"");
        }
    }
    if (attrs.getLength() > 0) nl();
    emit(">");
}

public void endElement(...)
throws SAXException
{
    nl();
    emit("END_ELM: ");
    emit("</"+name+">");
}

public void characters(char buf[], int offset, int Len)
throws SAXException
{
    nl(); emit("CHARS: |");
    String s = new String(buf, offset, Len);
    emit(s);
    emit("|");
}
```

Compile and run this version of the program to produce a more informative output listing. The attributes are now shown one per line, which is nice. But, more importantly, output lines like this one:

```
CHARS: |


   |
```

show that the `characters` method is responsible for echoing both the spaces that create the indentation and the multiple newlines that separate the attributes.

> **Note:** The XML specification requires all input line separators to be normalized to a single newline. The newline character is specified as `\n` in Java, C, and UNIX systems, but goes by the alias "linefeed" in Windows systems.

Compressing the Output

To make the output more readable, modify the program so that it only outputs characters containing something other than whitespace.

> **Note:** The code discussed in this section is in `Echo03.java`.

Make the changes shown below to suppress output of characters that are all whitespace:

```
public void characters(char buf[], int offset, int Len)
throws SAXException
{
    ~~nl(); emit("CHARS: |");~~
    nl(); emit("CHARS:   ");
    String s = new String(buf, offset, Len);
    ~~emit(s);~~
    ~~emit("|");~~
    if (!s.trim().equals("")) emit(s);
}
```

If you run the program now, you will see that you have eliminated the indentation as well, because the indent space is part of the whitespace that precedes the start of an element. Add the code highlighted below to manage the indentation:

```
static private Writer       out;

private String indentString = "    "; // Amount to indent
private int indentLevel = 0;

...

public void startElement(...)
throws SAXException
{
    indentLevel++;
    nl(); emit("ELEMENT: ");
    ...
}

public void endElement(...)
throws SAXException
{
    nl();
    emit("END_ELM: ");
    emit("</"+sName+">");
```

```
        indentLevel--;
    }
    ...
    private void nl()
    throws SAXException
    {
        ...
        try {
            out.write(lineEnd);
            for (int i=0; i < indentLevel; i++) out.write(indentString);

        } catch (IOException e) {
        ...
    }
```

This code sets up an indent string, keeps track of the current indent level, and outputs the indent string whenever the `nl` method is called. If you set the indent string to "", the output will be un-indented (Try it. You'll see why it's worth the work to add the indentation.)

You'll be happy to know that you have reached the end of the "mechanical" code you have to add to the Echo program. From here on, you'll be doing things that give you more insight into how the parser works. The steps you've taken so far, though, have given you a lot of insight into how the parser sees the XML data it processes. It's also given you a helpful debugging tool you can use to see what the parser sees.

Inspecting the Output

The complete output for this version of the program is shown in `Echo03-01`. Part of that output is shown here:

```
    ELEMENT: <slideshow
    ...
    CHARS:
    CHARS:
        ELEMENT: <slide
        ...
        END_ELM: </slide>
    CHARS:
    CHARS:
```

Note that the `characters` method was invoked twice in a row. Inspecting the source file `slideSample01.xml` shows that there is a comment before the first slide. The first call to `characters` comes before that comment. The second call comes after. (Later on, you'll see how to be notified when the parser encounters a comment, although in most cases you won't need such notifications.)

Note, too, that the `characters` method is invoked after the first slide element, as well as before. When you are thinking in terms of hierarchically structured data, that seems odd. After all, you intended for the `slideshow` element to contain `slide` elements, not text. Later on, you'll see how to restrict the `slideshow` element using a DTD. When you do that, the `characters` method will no longer be invoked.

In the absence of a DTD, though, the parser must assume that any element it sees contains text like that in the first item element of the overview slide:

```
     <item>Why <em>WonderWidgets</em> are great</item>
```

Here, the hierarchical structure looks like this:

```
     ELEMENT: <item>
     CHARS:   Why
         ELEMENT: <em>
         CHARS:   WonderWidgets
```

```
        END_ELM: </em>
    CHARS:    are great
    END_ELM: </item>
```

Documents and Data

In this example, it's clear that there are characters intermixed with the hierarchical structure of the elements. The fact that text can surround elements (or be prevented from doing so with a DTD or schema) helps to explain why you sometimes hear talk about "XML data" and other times hear about "XML documents". XML comfortably handles both structured data and text documents that include markup. The only difference between the two is whether or not text is allowed between the elements.

Note:
In an upcoming section of this tutorial, you will work with the `ignorableWhitespace` method in the `ContentHandler` interface. This method can only be invoked when a DTD is present. If a DTD specifies that `slideshow` does not contain text, then all of the whitespace surrounding the `slide` elements is by definition ignorable. On the other hand, if `slideshow` can contain text (which must be assumed to be true in the absence of a DTD), then the parser must assume that spaces and lines it sees between the `slide` elements are significant parts of the document.

7a. Defining Parameter Entities and Conditional Sections

Just as a general entity lets you reuse XML data in multiple places, a parameter entity lets you reuse parts of a DTD in multiple places. In this section of the tutorial, you 'll see how to define and use parameter entities. You'll also see how to use parameter entities with conditional sections in a DTD.

Link Summary
Exercise Links   • slideshow2.dtd   • slideshow2-dtd.html   • slideSample08.xml   • slideSample08-xml.html   • xhtml.dtd   ***External Links***   • Modularized XHTML   ***Glossary Terms***   content, DTD, general entity, mixed content model, namespace, parameter entity, SGML, URI, XHTML

Creating and Referencing a Parameter Entity

Recall that the existing version of the slide presentation could not be validated because the document used `<em>` tags, and those are not part of the DTD. In general, we'd like to use a whole variety of HTML-style tags in the text of a slide, not just one or two, so it makes more sense to use an existing DTD for XHTML than it does to define all the tags we might ever need. A parameter entity is intended for exactly that kind of purpose.

> **Note:**
> The DTD specifications shown here are contained in slideshow2.dtd. The XML file that references it is slideSample08.xml. (The browsable versions are slideshow2-dtd.html and slideSample08-xml.html.)

Open your DTD file for the slide presentation and add the text highlighted below to define a parameter entity that references an external DTD file:

```
<!ELEMENT slide (image?, title?, item*)>
<!ATTLIST slide
```

```
                 ...
    >

    <!ENTITY % xhtml SYSTEM "xhtml.dtd">
    %xhtml;

    <!ELEMENT title ...
```

Here, you used an `<!ENTITY>` tag to define a parameter entity, just as for a general entity, but using a somewhat different syntax. You included a percent sign (%) before the entity name when you defined the entity, and you used the percent sign instead of an ampersand when you referenced it.

Also, note that there are always two steps for using a parameter entity. The first is to define the entity name. The second is to reference the entity name, which actually does the work of including the external definitions in the current DTD. Since the URI for an external entity could contain slashes (/) or other characters that are not valid in an XML name, the definition step allows a valid XML name to be associated with an actual document. (This same technique is used in the definition of namespaces, and anywhere else that XML constructs need to reference external documents.)

Notes:

- The DTD file referenced by this definition is `xhtml.dtd`. You can either copy that file to your system or modify the `SYSTEM` identifier in the `<!ENTITY>` tag to point to the correct URL.
- This file is a small subset of the XHTML specification, loosely modeled after the Modularized XHTML draft, which aims at breaking up the DTD for XHTML into bite-sized chunks, which can then be combined to create different XHTML subsets for different purposes. When work on the modularized XHTML draft has been completed, this version of the DTD should be replaced with something better. For now, this version will suffice for our purposes.

The whole point of using an XHTML-based DTD was to gain access to an entity it defines that covers HTML-style tags like `<em>` and `<b>`. Looking through `xhtml.dtd` reveals the following entity, which does exactly what we want:

```
    <!ENTITY % inline "#PCDATA|em|b|a|img|br">
```

This entity is a simpler version of those defined in the Modularized XHTML draft. It defines the HTML-style tags we are most likely to want to use -- emphasis, bold, and break, plus a couple of others for images and anchors that we may or may not use in a slide presentation. To use the `inline` entity, make the changes highlighted below in your DTD file:

```
    <!ELEMENT title (~~#PCDATA~~ %inline;)*>
```

```
<!ELEMENT item (#PCDATA %inline; | item)* >
```

These changes replaced the simple #PCDATA item with the inline entity. It is important to notice that #PCDATA is first in the inline entity, and that inline is first wherever we use it. That is required by XML's definition of a mixed-content model. To be in accord with that model, you also had to add an asterisk at the end of the title definition. (In the next two sections, you'll see that our definition of the title element actually conflicts with a version defined in xhtml.dtd, and see different ways to resolve the problem.)

Note:
The Modularized XHTML DTD defines both inline and Inline entities, and does so somewhat differently. Rather than specifying #PCDATA|em|b|a|img|Br, their definitions are more like (#PCDATA|em|b|a|img|Br)*. Using one of those definitions, therefore, looks more like this:

```
<!ELEMENT title %Inline; >
```

Conditional Sections

Before we proceed with the next programming exercise, it is worth mentioning the use of parameter entities to control *conditional sections*. Although you cannot conditionalize the content of an XML document, you can define conditional sections in a DTD that become part of the DTD only if you specify include. If you specify ignore, on the other hand, then the conditional section is not included.

Suppose, for example, that you wanted to use slightly different versions of a DTD, depending on whether you were treating the document as an XML document or as a SGML document. You could do that with DTD definitions like the following:

```
someExternal.dtd:
    <![ INCLUDE [
        ... XML-only definitions
    ]]>
    <![ IGNORE [
        ... SGML-only definitions
    ]]>
    ... common definitions
```

The conditional sections are introduced by "<![", followed by the INCLUDE or IGNORE keyword and another "[". After that comes the contents of the conditional section, followed by the terminator: "]]>". In this case, the XML definitions are included, and the SGML definitions are excluded. That's fine for XML documents, but you can't use the DTD for SGML documents. You could change the keywords, of course, but that only reverses the problem.

The solution is to use references to parameter entities in place of the `INCLUDE` and `IGNORE` keywords:

```
someExternal.dtd:
    <![ %XML; [
        ... XML-only definitions
    ]]>
    <![ %SGML; [
        ... SGML-only definitions
    ]]>
    ... common definitions
```

Then each document that uses the DTD can set up the appropriate entity definitions:

```
<!DOCTYPE foo SYSTEM "someExternal.dtd" [
    <!ENTITY % XML  "INCLUDE" >
    <!ENTITY % SGML "IGNORE" >
]>
<foo>
  ...
</foo>
```

This procedure puts each document in control of the DTD. It also replaces the `INCLUDE` and `IGNORE` keywords with variable names that more accurately reflect the purpose of the conditional section, producing a more readable, self-documenting version of the DTD.

2b. Adding Additional Event Handlers

Besides `ignorableWhitespace`, there are two other `ContentHandler` methods that can find uses in even simple applications: `setDocumentLocator` and `processingInstruction`. In this section of the tutorial, you'll implement those two event handlers.

Link Summary

Local Links

- A Quick Introduction to XML

Examples

- Echo04.java
- Echo04-01
- slideSample02.xml
- slideSample02-xml.html
- Echo05.java
- Echo05-02

API References

- Locator

Glossary Terms

URL, URN

Identifying the Document's Location

A *locator* is an object that contains the information necessary to find the document. The `Locator` class encapsulates a system ID (URL) or a public identifier (URN), or both. You would need that information if you wanted to find something relative to the current document -- in the same way, for example, that an HTML browser processes an `href="anotherFile"` attribute in an anchor tag -- the browser uses the location of the current document to find `anotherFile`.

You could also use the locator to print out good diagnostic messages. In addition to the document's location and public identifier, the locator contains methods that give the column and line number of the most recently-processed event. The `setDocumentLocator` method is called only once at the beginning of the parse, though. To get the current line or column number, you would save the locator when `setDocumentLocator` is invoked and then use it in the other event-handling methods.

> **Note:**
> The code discussed in this section is in `Echo04.java`. Its output is stored at `Echo04-01`.

Add the method below to the Echo program to get the document locator and use it to echo the document's system ID.

```
...
private String indentString = "    "; // Amount to indent
private int indentLevel = 0;

public void setDocumentLocator(Locator l)
{
    try {
      out.write("LOCATOR");
      out.write("\n SYS ID: " + l.getSystemId() );
      out.flush();
    } catch (IOException e) {
        // Ignore errors
    }
}

public void startDocument()
...
```

Notes:

- This method, in contrast to every other `ContentHandler` method, does not return a `SAXException`. So, rather than using `emit` for output, this code writes directly to `System.out`. (This method is generally expected to simply save the `Locator` for later use, rather than do the kind of processing that generates an exception, as here.)
- The spelling of these methods is "Id", not "ID". So you have `getSystemId` and `getPublicId`.

When you compile and run the program on `slideSample01.xml`, here is the significant part of the output:

```
LOCATOR
 SYS ID: file:<path>/../samples/slideSample01.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
...
```

Here, it is apparent that `setDocumentLocator` is called before `startDocument`. That can make a difference if you do any initialization in the event handling code.

Handling Processing Instructions

It sometimes makes sense to code application-specific processing instructions in the XML data. In this exercise, you'll add a processing instruction to your `slideSample.xml` file and then modify the Echo program to display it.

> **Note:**
> The code discussed in this section is in `Echo05.java`. The file it operates on is `slideSample02.xml`. (The browsable version is slideSample02-xml.html.) The output is stored at `Echo05-02`.

As you saw in A Quick Introduction to XML, the format for a processing instruction is `<?target data?>`, where "target" is the target application that is expected to do the processing, and "data" is the instruction or information for it to process. Add the text highlighted below to add a processing instruction for a mythical slide presentation program that will query the user to find out which slides to display (technical, executive-level, or all):

```
<slideshow
    ...
    >

    <!-- PROCESSING INSTRUCTION -->
    <?my.presentation.Program QUERY="exec, tech, all"?>

    <!-- TITLE SLIDE -->
```

Notes:

- The "data" portion of the processing instruction can contain spaces, or may even be null. But there cannot be any space between the initial <? and the target identifier.
- The data begins after the first space.
- Fully qualifying the target with the complete web-unique package prefix makes sense, so as to preclude any conflict with other programs that might process the same data.
- For readability, it seems like a good idea to include a colon (:) after the name of the application, like this:
  `<?my.presentation.Program: QUERY="..."?>`
  The colon makes the target name into a kind of "label" that identifies the intended recipient of the

instruction. However, while the w3c spec allows ":" in a target name, some versions of IE5 consider it an error. For this tutorial, then, we avoid using a colon in the target name.

Now that you have a processing instruction to work with, add the code highlighted below to the Echo app:

```
public void characters(char buf[], int offset, int len)
...
}

public void processingInstruction(String target, String data)
throws SAXException
{
    nl();
    emit("PROCESS: ");
    emit("<?"+target+" "+data+"?>");
}

private void emit(String s)
...
```

When your edits are complete, compile and run the program. The relevant part of the output should look like this:

```
...
CHARS:
CHARS:
PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
CHARS:
CHARS:
...
```

Now that you've had a chance to work with the processing instruction, you can remove that instruction from the XML file. You won't be needing it any more.

Summary

With the minor exception of `ignorableWhitespace`, you have used most of the `ContentHandler` methods that you need to handle the most commonly useful SAX events. You'll see `ignorableWhitespace` a little later on. Next, though, you'll get deeper insight into how you handle errors in the SAX parsing process.

3. Handling Errors with the Nonvalidating Parser

This version of the Echo program uses the nonvalidating parser. So it can't tell if the XML document contains the right tags, or if those tags are in the right sequence. In other words, it can't tell you if the document is valid. It can, however, tell whether or not the document is well-formed.

In this section of the tutorial, you'll modify the slideshow file to generate different kinds of errors and see how the parser handles them. You'll also find out which error conditions are ignored, by default, and see how to handle them.

Link Summary

Exercises

- slideSampleBad1.xml
- slideSampleBad1-xml.html
- Echo05-Bad1
- Echo06.java
- Echo06-Bad1
- slideSampleBad2.xml
- slideSampleBad2-xml.html
- Echo06-Bad2
- Echo07.java
- Echo07-Bad2

API Links

- ContentHandler
- ErrorHandler
- DefaultHandler

Glossary Terms

DTD, error, fatal error, valid, warning, well-formed

Introducing an Error

The parser can generate one of three kinds of errors: fatal error, error, and warning. In this exercise, you'll make a simple modification to the XML file to introduce a fatal error. Then you'll see how it's handled in the Echo app.

> **Note:** The XML structure you'll create in this exercise is in slideSampleBad1.xml. (The browsable version is slideSampleBad1-xml.html.) The output is in Echo05-Bad1.

One easy way to introduce a fatal error is to remove the final "/" from the empty `item` element to create a tag that does not have a corresponding end tag. That constitutes a fatal error, because all XML documents must, by definition, be well formed. Do the following:

1. Copy `slideSample.xml` to `badSample.xml`.
2. Edit `badSample.xml` and remove the character shown below:

```
 ...
<!-- OVERVIEW -->
<slide type="all">
  <title>Overview</title>
  <item>Why <em>WonderWidgets</em> are great</item>
  <item~~/~~>
  <item>Who <em>buys</em> WonderWidgets</item>
</slide>
```

```
 ...
```

to produce:

```
 ...
<item>Why <em>WonderWidgets</em> are great</item>
<item>
<item>Who <em>buys</em> WonderWidgets</item>
 ...
```

3. Run the Echo program on the new file.

The output you get now looks like this:

```
...
        ELEMENT: <item>
        CHARS:   The
            ELEMENT: <em>
            CHARS:   Only
            END_ELM: </em>
        CHARS:    Section
        END_ELM: </item>
    CHARS:
    END_ELM:
CHARS:   org.xml.sax.SAXParseException: Expected "</item>"
         to terminate element starting on line 20.
...
at javax.xml.parsers.SAXParser.parse(SAXParser.java:286)
at Echo05.main(Echo05.java:61)
```

When a fatal error occurs, the parser is unable to continue. So, if the application does not generate an exception (which you'll see how to do a moment), then the default error-event handler generates one. The stack trace is generated by the `Throwable` exception handler in your main method:

```
  ...
} catch (Throwable t) {
    t.printStackTrace();
}
```

That stack trace is not too useful, though. Next, you'll see how to generate better diagnostics when an error occurs.

Handling a SAXParseException

When the error was encountered, the parser generated a `SAXParseException` -- a subclass of `SAXException` that identifies the file and location where the error occurred.

Note: The code you'll create in this exercise is in `Echo06.java`. The output is in `Echo06-Bad1`.

Add the code highlighted below to generate a better diagnostic message when the exception occurs:

```
  ...
} catch (SAXParseException spe) {
   // Error generated by the parser
   System.out.println("\n** Parsing error"
      + ", line " + spe.getLineNumber()
      + ", uri " + spe.getSystemId());
   System.out.println("   " + spe.getMessage() );

} catch (Throwable t) {
    t.printStackTrace();
}
```

Running the program now generates an error message which is a bit more helpful, like this:

```
** Parsing error, line 22, uri file:<path>/slideSampleBad1.xml
   Next character must be...
```

Note:
Catching all throwables like this is *not* a good idea for production applications. We're just doing it now so we can build up to full error handling gradually.

Handling a SAXException

A more general `SAXException` instance may sometimes be generated by the parser, but it more frequently occurs when an error originates in one of application's event handling methods. For example, the signature of the `startDocument` method in the `ContentHandler` interface is defined as returning a `SAXException`:

```
public void startDocument() throws SAXException
```

All of the `ContentHandler` methods (except for `setDocumentLocator`) have that signature declaration.

A `SAXException` can be constructed using a message, another exception, or both. So, for example, when `Echo.startDocument` outputs a string using the `emit` method, any I/O exception that occurs is wrapped in a `SAXException` and sent back to the parser:

```
private void emit(String s)
throws SAXException
{
    try {
        out.write(s);
        out.flush();
    } catch (IOException e) {
        throw new SAXException("I/O error", e);
    }
}
```

Note: If you saved the `Locator` object when `setDocumentLocator` was invoked, you could use it to generate a `SAXParseException`, identifying the document and location, instead of generating a

SAXException.

When the parser delivers the exception back to the code that invoked the parser, it makes sense to use the original exception to generate the stack trace. Add the code highlighted below to do that:

```
  ...
} catch (SAXParseException err) {
    System.out.println("** Parsing error"
        + ", line " + err.getLineNumber()
        + ", uri " + err.getSystemId());
    System.out.println("   " + err.getMessage());

} catch (SAXException sxe) {
    // Error generated by this application
    // (or a parser-initialization error)
    Exception  x = sxe;
    if (sxe.getException() != null)
        x = sxe.getException();
    x.printStackTrace();

} catch (Throwable t) {
    t.printStackTrace();
}
```

This code tests to see if the SAXException is wrapping another exception. If so, it generates a stack trace originating from where that exception occurred to make it easier to pinpoint the code responsible for the error. If the exception contains only a message, the code prints the stack trace starting from the location where the exception was generated.

Improving the SAXParseException Handler

Since the SAXParseException can also wrap another exception, add the code highlighted below to use it for the stack trace:

```
   ...
} catch (SAXParseException err) {
     System.out.println("** Parsing error"
        + ", line " + err.getLineNumber()
        + ", uri " + err.getSystemId());
     System.out.println("   " + err.getMessage());

     // Unpack the delivered exception to get the exception it contains
     Exception  x = spe;
       if (spe.getException() != null)
           x = spe.getException();
       x.printStackTrace();

} catch (SAXException e) {
    // Error generated by this application
    // (or a parser-initialization error)
    Exception   x = e;
```

```
        if (e.getException() != null)
            x = e.getException();
        x.printStackTrace();

    } catch (Throwable t) {
        t.printStackTrace();
    }
```

The program is now ready to handle any SAX parsing exceptions it sees. You've seen that the parser generates exceptions for fatal errors. But for nonfatal errors and warnings, exceptions are never generated by the default error handler, and no messages are displayed. Next, you'll learn more about errors and warnings and find out how to supply an error handler to process them.

Handling a ParserConfigurationException

Finally, recall that the SAXParserFactory class could throw an exception if it were for unable to create a parser. Such an error might occur if the factory could not find the class needed to create the parser (class not found error), was not permitted to access it (illegal access exception), or could not instantiate it (instantiation error).

Add the code highlighted below to handle such errors:

```
    } catch (SAXException e) {
        Exception   x = e;
        if (e.getException() != null)
            x = e.getException();
        x.printStackTrace();

    } catch (ParserConfigurationException pce) {
        // Parser with specified options can't be built
        pce.printStackTrace();

    } catch (Throwable t) {
        t.printStackTrace();
```

This code, like the SAXException handler, takes into account the possibility that the reported exception might be wrapping another exception. (Admittedly, there are quite a few error handlers here. But at least now you know the kinds of exceptions that can occur.)

> **Note:**
> A `javax.xml.parsers.`FactoryConfigurationError could also be thrown if the factory class specified by the system property cannot be found or instantiated. That is a non-trappable error, since the program is not expected to be able to recover from it.

Handling an IOException

Finally, while we're at it, let's stop intercepting all Throwable objects and catch the only remaining exceptions there is to catch, IOExceptions:

```
    } catch (ParserConfigurationException pce) {
        // Parser with specified options can't be built
        pce.printStackTrace();

    } catch (Throwable t) {
        t.printStackTrace();
    } catch (IOException ioe) {
        // I/O error
        ioe.printStackTrace();
    }
```

Understanding NonFatal Errors

In general, a nonfatal *error* occurs when an XML document fails a validity constraint. If the parser finds that the document is not valid (which means that it contains an invalid tag or a tag in location that is disallowed), then an error event is generated. In general, then, errors are generated by a validaating parser, given a DTD that tells it which tags are valid. There is one kind of error, though, that is generated by the nonvalidating parser you have been working with so far. You'll experiment with that error next.

> **Note:** The file you'll create in this exercise is `slideSampleBad2.xml`. (The browsable version is slideSampleBad2-xml.html.) The output is in `Echo06-Bad2`.

The SAX specification requires an error event to be generated if the XML document uses a version of XML that the parser does not support. To generate such an error, make the changes shown below to alter your XML file so it specifies `version="1.2"`.

```
    <?xml version='1.~~0~~2' encoding='utf-8'?>
```

Now run your version of the Echo program on that file. What happens? (See below for the answer.)

> **Answer:** Nothing happens! By default, the error is ignored. The output from the Echo program looks the same as if `version="1.0"` had been properly specified. To do something else, you need to supply your own error handler. You'll do that next.

Handling Nonfatal Errors

A standard treatment for "nonfatal" errors is to treat them as if they were fatal. After all, if a validation error occurs in a document you are processing, you probably don't want to continue processing it. In this exercise, you'll do exactly that.

> **Note:** The code for the program you'll create in this exercise is in `Echo07.java`. The output is in `Echo07-Bad2`.

To take over error handling, you override the DefaultHandler methods that handle fatal errors, nonfatal errors, and warnings as part of the `ErrorHandler` interface. The SAX parser delivers a `SAXParseException` to each of these methods, so generating an exception when an error occurs is as simple as throwing it back.

Add the code highlighted below to override the handlers for errors:

```
    public void processingInstruction(String target, String data)
```

```
throws SAXException
{
  nl();
  emit("PROCESS: ");
  emit("<?"+target+" "+data+"?>");
}

// treat validation errors as fatal
public void error(SAXParseException e)
throws SAXParseException
{
  throw e;
}
```

Now when you run your app on the file with the faulty version number, you get an exception, as shown here (but slightly reformatted for readability):

```
START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
   ** Parsing error, line 1, uri file:/<path>/slideSampleBad2.xml
   XML version "1.0" is recognized, but not "1.2".
org.xml.sax.SAXParseException: XML version "1.0" is recognized, but not "1.2".
...
at javax.xml.parsers.SAXParser.parse(SAXParser.java:286)
at Echo07.main(Echo07.java:61)
```

Note: The error actually occurs after the `startDocument` event has been generated. The document header that the program "echoes" is the one it creates on the assumption that everything is ok, rather than the one that is actually in the file.

Handling Warnings

Warnings, too, are ignored by default. Warnings are informative, and require a DTD. For example, if an element is defined twice in a DTD, a warning is generated -- it's not illegal, and it doesn't cause problems, but it's something you might like to know about since it might not have been intentional.

Add the code highlighted below to generate a message when a warning occurs:

```
// treat validation errors as fatal
public void error(SAXParseException e)
throws SAXParseException
{
  throw e;
}

// dump warnings too
public void warning(SAXParseException err)
throws SAXParseException
{
  System.out.println("** Warning"
      + ", line " + err.getLineNumber()
      + ", uri " + err.getSystemId());
  System.out.println("   " + err.getMessage());
}
```

Since there is no good way to generate a warning without a DTD, you won't be seeing any just yet. But when one does occur, you're ready!

> **Note:** By default, `DefaultHandler` throws an exception when a fatal error occurs. You could override the `fatalError` method to throw a different exception, if you like. But if your code doesn't, the reference implementation's SAX parser will.

7b. Parsing the Parameterized DTD

This section uses the Echo program to see what happens when you reference `xhtml.dtd` in `slideshow.dtd`. It also covers the kinds of warnings that are generated by the SAX parser when a DTD is present.

> Note: The output described in this section is contained in `Echo10-08`.

When you try to echo the slide presentation, you find that it now contains a new error. The relevant part of the output is shown here (formatted for readability):

```
<?xml version='1.0' encoding='UTF-8'?>
** Parsing error, line 22,
   uri file:.../slideshow.dtd
Element "title" was already declared.
org.xml.sax.SAXParseException: ...
```

It seems that `xhtml.dtd` defines a `title` element which is entirely different from the `title` element defined in the slideshow DTD. Because there is no hierarchy in the DTD, these two definitions conflict.

> **Note:**
> The Modularized XHTML DTD also defines a `title` element that is intended to be the document title, so we can't avoid the conflict by changing `xhtml.dtd` -- the problem would only come back to haunt us later.

You could also use XML namespaces to resolve the conflict, or use one of the more hierarchical schema proposals

Link Summary

Local Links

- Schema Proposals
- Manipulating Document Contents with the Document Object Model

Exercise Links

- Echo10-08
- slideshow3.dtd
- slideshow3-dtd.html
- slideSample09.xml
- slideSample09-xml.html
- copyright.xml
- copyright-xml.html
- xhtml.dtd
- xhtml-dtd.html
- Echo10-09

External Links

- Modularized XHTML

Glossary Terms

namespace

described in Schema Proposals. For now, though, let's simply rename the `title` element in `slideshow.dtd`.

> **Note:**
> The XML shown here is contained in slideshow3.dtd and slideSample09.xml, which references copyright.xml and xhtml.dtd. (The browsable versions are slideshow3-dtd.html, slideSample09-xml.html, copyright-xml.html, and xhtml-dtd.html.) The results of processing are shown in Echo10-09.

To keep the two title elements separate, we'll resort to a "hyphenation hierarchy". Make the changes highlighted below to change the name of the `title` element in `slideshow.dtd` to `slide-title`:

```
<!ELEMENT slide (image?, slide-title?, item*)>
<!ATTLIST slide
            type   (tech | exec | all) #IMPLIED
>

<!-- Defines the %inline; declaration -->
<!ENTITY % xhtml SYSTEM "xhtml.dtd">
%xhtml;

<!ELEMENT slide-title (%inline;)*>
```

The next step is to modify the XML file to use the new element name. To do that, make the changes highlighted below:

```
...
<slide type="all">
<slide-title>Wake up to ... </slide-title>
</slide>

...

<!-- OVERVIEW -->
<slide type="all">
<slide-title>Overview</slide-title>
<item>...
```

Now run the Echo program on this version of the slide presentation. It should run to completion and display output like that shown in `Echo10-09`.

Congratulations! You have now read a fully validated XML document. The changes you made had the effect of putting your DTD's `title` element into a slideshow "namespace" that you artificially constructed by hyphenating the name. Now the `title` element in the "slideshow namespace" (`slide-title`, really) no longer conflicts with the `title` element in `xhtml.dtd`. In the next section of the tutorial, you'll see how to do that without renaming the definition. To finish off this section, we'll take a look at the kinds of warnings that the validating parser can produce when processing the DTD.

DTD Warnings

As mentioned earlier in this tutorial, warnings are generated only when the SAX parser is processing a DTD. Some warnings are generated only by the validating parser. The nonvalidating parser's main goal is operate as rapidly as possible, but it too generates some warnings. (The explanations that follow tell which does what.)

The XML specification suggests that warnings should be generated as result of:

- Providing additional declarations for entities, attributes, or notations.
  (Such declarations are ignored. Only the first is used. Also, note that duplicate definitions of *elements* always produce a fatal error when validating, as you saw earlier.)
- Referencing an undeclared element type.
  (A validity error occurs only if the undeclared type is actually used in the XML document. A warning results when the undeclared element is referenced in the DTD.)
- Declaring attributes for undeclared element types.

The Java XML SAX parser also emits warnings in other cases, such as:

- No <!DOCTYPE ...> when validating.
- Referencing an undefined parameter entity when not validating.
  (When validating, an error results. Although nonvalidating parsers are not required to read parameter entities, the Java XML parser does so. Since it is not a requirement, the Java XML parser generates a warning, rather than an error.)
- Certain cases where the character-encoding declaration does not look right.

At this point, you have digested many XML concepts, including DTDs, external entities. You have also learned your way around the SAX parser. The remainder of the SAX tutorial covers advanced topics that you will only need to understand if you are writing SAX-based applications. If your primary goal is to write DOM-based apps, you can skip ahead to Manipulating Document Contents with the Document Object Model.

9. Using the DTDHandler and EntityResolver

In this section of the tutorial, we'll carry on a short discussion of the two remaining SAX event handlers: `DTDHandler` and `EntityResolver`. The `DTDHandler` is invoked when the DTD encounters an unparsed entity or a notation declaration. The `EntityResolver` comes into play when a URN (public ID) must be resolved to a URL (system ID).

Link Summary

Local Links

- Referencing Binary Entities

API Links

- DefaultHandler
- DTDHandler
- EntityResolver
- InputSource

Glossary Terms

notation, SGML, unparsed entity, URL, URN

The DTDHandler API

In the section Referencing Binary Entities you saw a method for referencing a file that contains binary data, like an image file, using MIME data types. That is the simplest, most extensible mechanism to use. For compatibility with older SGML-style data, though, it is also possible to define an unparsed entity.

The `NDATA` keyword defines an unparsed entity, like this:

```
    <!ENTITY myEntity SYSTEM "..URL.." NDATA gif>
```

The `NDATA` keyword says that the data in this entity is not parsable XML data, but is instead data that uses some other notation. In this case, the notation is named "gif". The DTD must then include a declaration for that notation, which would look something like this:

```
    <!NOTATION gif SYSTEM "..URL..">
```

When the parser sees an unparsed entity or a notation declaration, it does nothing with the information except to pass it along to the application using the `DTDHandler` interface. That interface defines two methods:

```
notationDecl(String name, String publicId, String systemId)

unparsedEntityDecl(String name, String publicId, String systemId,
                   String notationName)
```

The `notationDecl` method is passed the name of the notation and either the public or system identifier, or both, depending on which is declared in the DTD. The `unparsedEntityDecl` method is passed the name of the entity, the appropriate identifiers, and the name of the notation it uses.

> **Note:**
> The DTDHandler interface is implemented by the DefaultHandler class.

Notations can also be used in attribute declarations. For example, the following declaration requires notations for the GIF and PNG image-file formats:

```
<!ENTITY image EMPTY>
<!ATTLIST image
            ...
            type  NOTATION  (gif | png) "gif"
>
```

Here, the `type` is declared as being either `gif`, or `png`. The default, if neither is specified, is `gif`.

Whether the notation reference is used to describe an unparsed entity or an attribute, it is up to the application to do the appropriate processing. The parser knows nothing at all about the semantics of the notations. It only passes on the declarations.

The EnityResolver API

The `EntityResolver` API lets you convert a public ID (URN) into a system ID (URL). Your application may need to do that, for example, to convert something like `href="urn:/someName"` into `"http://someURL"`.

The `EntityResolver` interface defines a single method:

```
resolveEntity(String publicId, String systemId)
```

This method returns an `InputSource` object, which can be used to access the entity's contents. Converting an URL into an `InputSource` is easy enough. But the URL that is passed as the system ID will be the location of the original document which is, as likely as not, somewhere out on the Web. To access a local copy, if there is one, you must maintain a catalog somewhere on the system that maps names (public IDs) into local URLs.

Java's Encoding Schemes

This sidebar describes the character-encoding schemes that are supported by the Java platform. Use your browser's back button to continue in the document that brought you here.

US-ASCII

US-ASCII is a 7-bit encoding scheme that covers the English-language alphabet. It is not large enough to cover the characters used in other languages, however, so it is not very useful for internationalization.

UTF-8

UTF-8 is an 8-bit encoding scheme. Characters from the English-language alphabet are all encoded using an 8-bit bytes. Characters for other languages are encoding using 2, 3 or even 4 bytes. UTF-8 therefore produces compact documents for the English language, but very large documents for other languages. If the majority of a document's text is in English, then UTF-8 is a good choice because it allows for internationalization while still minimizing the space required for encoding.

UTF-16

UTF-16 is a 16-bit encoding scheme. It is large enough to encode all the characters from all the alphabets in the world, with the exception of ideogram-based languages like Chinese. All characters in UTF-16 are encoded using 2 bytes. An English-language document that uses UTF-16 will be twice as large as the same document encoded using UTF-8. Documents written in other languages, however, will be far smaller using UTF-16.

5d. Referencing Binary Entities

This section contains no programming exercises. Instead, it discusses the options for referencing binary files like image files and multimedia data files.

Link Summary

API Links

- DTDHandler

External Links

- HTML 4.0 Specification
- MIME data types

Glossary Terms

namespace, `NOTATION`, SGML

Using a MIME Data Type

There are two ways to go about referencing an unparsed entity like a binary image file. One is to use the DTD's `NOTATION`-specification mechanism. However, that mechanism is a complex, non-intuitive holdover that mostly exists for compatibility with SGML documents. We will have occasion to discuss it in a bit more depth when we look at the DTDHandler API, but suffice it for now to say that the combination of the recently defined XML namespaces standard, in conjunction with the MIME data types defined for electronic messaging attachments, together provide a much more useful, understandable, and extensible mechanism for referencing unparsed external entities.

> **Note:** The XML described here is in `slideshow1b.dtd`. We won't actually be echoing any images. That's beyond the scope of this tutorial's Echo program. This section is simply for understanding how such references can be made. It assumes that the application which will be processing the XML data knows how to handle such references.

To set up the slideshow to use image files, add the text highlighted below to your `slideshow.dtd` file:

```
<!ELEMENT slide (image?, title, item*)>
<!ATTLIST slide
            type   (tech | exec | all) #IMPLIED
>
<!ELEMENT title (#PCDATA)>
<!ELEMENT item (#PCDATA | item)* >
<!ELEMENT image EMPTY>
<!ATTLIST image
```

```
                alt     CDATA    #IMPLIED
                src     CDATA    #REQUIRED
                type    CDATA    "image/gif"
    >
```

These modifications declare `image` as an optional element in a `slide`, define it as empty element, and define the attributes it requires. The `image` tag is patterned after the HTML 4.0 tag, `img`, with the addition of an image-type specifier, `type`. (The `img` tag is defined in the HTML 4.0 Specification.)

The `image` tag's attributes are defined by the `ATTLIST` entry. The `alt` attribute, which defines alternate text to display in case the image can't be found, accepts character data (`CDATA`). It has an "implied" value, which means that it is optional, and that the program processing the data knows enough to substitute something like "Image not found". On the other hand, the `src` attribute, which names the image to display, is required.

The `type` attribute is intended for the specification of a MIME data type, as defined at `ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/`. It has a default value: `image/gif`.

> **Note:** It is understood here that the character data (`CDATA`) used for the type attribute will be one of the MIME data types. The two most common formats are: `image/gif`, and `image/jpeg`. Given that fact, it might be nice to specify an attribute list here, using something like:
>
> ```
>    type ("image/gif", "image/jpeg")
> ```
>
> That won't work, however, because attribute lists are restricted to name tokens. The forward slash isn't part of the valid set of name-token characters, so this declaration fails. Besides that, creating an attribute list in the DTD would limit the valid MIME types to those defined today. Leaving it as `CDATA` leaves things more open ended, so that the declaration will continue to be valid as additional types are defined.

In the document, a reference to an image named "intro-pic" might look something like this:

```
    <image src="image/intro-pic.gif", alt="Intro Pic", type="image/gif" />
```

The Alternative: Using Entity References

Using a MIME data type as an attribute of an element is a mechanism that is flexible and expandable. To create an external `ENTITY` reference using the notation mechanism, you need DTD `NOTATION` elements for jpeg and gif data. Those can of course be obtained from some central repository. But then you need to define a different `ENTITY` element for each image you intend to reference! In other words, adding a new image to your document always requires both a new entity definition in the DTD and a reference to it in the document. Given the anticipated ubiquity of the HTML 4.0 specification, the newer standard is to use the MIME data types and a declaration like `image`, which assumes the application knows how to process such elements.

Top *Contents* *Index* *Glossary*

2. Writing Out a DOM as an XML File

Once you have constructed a DOM, either by parsing an XML file or building it programmatically, you frequently want to save it as XML. This section shows you how to do that using the XSLT transform package.

Using that package, you'll create a transformer object to wire a DomSource to a StreamResult. You'll then invoke the transformer's `transform()` method to do the job!

Link Summary

Local Links

- Reading XML into a DOM, Additional Information
- Compiling the Program
- Running the Program

Exercise Links

- slideSample01.xml
- slideSample01-xml.html
- TransformationApp01.java
- TransformationApp02.java
- TransformationApp03.java
- TransformationLog02
- TransformationLog03

API Links

- DomSource
- StreamResult

Reading the XML

The first step is to create a DOM in memory by parsing an XML file. By now, you should be getting pretty comfortable with the process!

> **Note:**
> The code discussed in this section is in `TransformationApp01.java`.

The code below provides a basic template to start from. (It should be familiar. It's basically the same code you wrote at the start of the DOM tutorial. If you saved it then, that version should be pretty much the equivalent of what you see below.)

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

import java.io.*;

public class TransformationApp
{
    static Document document;

    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println ("Usage: java TransformationApp filename");
            System.exit (1);
        }

        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        //factory.setNamespaceAware(true);
        //factory.setValidating(true);

        try {
            File f = new File(argv[0]);
            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(f);

         } catch (SAXException sxe) {
           // Error generated by this application
           // (or a parser-initialization error)
           Exception  x = sxe;
           if (sxe.getException() != null)
               x = sxe.getException();
           x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }

    } // main

}
```

Creating a Transformer

The next step is to create a transformer you can use to transmit the XML to System.out.

> **Note:**
> The code discussed in this section is in TransformationApp02.java. The file it runs on is slideSample01.xml. (The browsable version is slideSample01-xml.html.) The output is in TransformationLog02.

Start by adding the import statements highlighted below:

```
    import javax.xml.transform.Transformer;
    import javax.xml.transform.TransformerFactory;
    import javax.xml.transform.TransformerException;
    import javax.xml.transform.TransformerConfigurationException;

    import javax.xml.transform.dom.DOMSource;

    import javax.xml.transform.stream.StreamResult;

    import java.io.*;
```

Here, you've added a series of classes which should be now be forming a standard pattern: an entity (Transformer), the factory to create it (TransformerFactory), and the exceptions that can be generated by each. Since a transformation always has a *source* and a *result*, you then imported the classes necessary to use a DOM as a source (DomSource), and an output stream for the the result (StreamResult).

Next, add the code to carry out the transformation:

```
try {
   File f = new File(argv[0]);
   DocumentBuilder builder = factory.newDocumentBuilder();
   document = builder.parse(f);

   // Use a Transformer for output
   TransformerFactory tFactory =
       TransformerFactory.newInstance();
   Transformer transformer = tFactory.newTransformer();

   DOMSource source = new DOMSource(document);
   StreamResult result = new StreamResult(System.out);
   transformer.transform(source, result);
```

Here, you created a transformer object, used the DOM to construct a source object, and used System.out to construct a result object. You then told the transformer to operate on the source object and output to the result object.

Note:
In this case, the "transformer" isn't actually changing anything. In XSLT terminology, you are using the *identity transform*, which means that the "transformation" generates a copy of the source, unchanged..

Finally, add the code highlighted below to catch the new errors that can be generated:

```
} catch (TransformerConfigurationException tce) {
   // Error generated by the parser
   System.out.println ("\n** Transformer Factory error");
   System.out.println("   " + tce.getMessage() );

   // Use the contained exception, if any
   Throwable x = tce;
   if (tce.getException() != null)
       x = tce.getException();
   x.printStackTrace();

} catch (TransformerException te) {
   // Error generated by the parser
   System.out.println ("\n** Transformation error");
   System.out.println("   " + te.getMessage() );

   // Use the contained exception, if any
   Throwable x = te;
   if (te.getException() != null)
       x = te.getException();
   x.printStackTrace();

} catch (SAXException sxe) {
   ...
```

Notes:

- TransformerExceptions are thrown by the transformer object.
- TransformerConfigurationExceptions are thrown by the factory.

Addendum:
Astute reader Malcolm Gorman points out that, as it is currently written, the transformation app won't preserve athe XML document's `DOCTYPE` setting. He proposes the following code to remedy the omission:

```
String systemValue = (new File(document.getDoctype().getSystemId())).getName();
transformer.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM, systemValue);
```

Writing the XML

For instructions on how to compile and run the program, see Compiling the Program and Running the Program, from the SAX tutorial. (Substitute "TransformationApp" for "Echo" as the name of the program.)

When you run the program on `slideSample01.xml`, this is the output you see:

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

<!--  A SAMPLE set of slides  -->
<slideshow title="Sample Slide Show" date="Date of publication" author="Yours Truly">


  <!-- TITLE SLIDE -->

  <slide type="all">

    <title>Wake up to WonderWidgets!</title>

  </slide>


  <!-- OVERVIEW -->

  <slide type="all">

    <title>Overview</title>

    <item>Why
      <em>WonderWidgets</em> are great
    </item>

    <item />

    <item>Who
      <em>buys</em> WonderWidgets
    </item>

  </slide>


</slideshow>
```

Note:
See Reading XML into a DOM, Additional Information to find out more about configuring the factory and handling validation errors.

Writing Out a Subtree of the DOM

It is also possible to operate on a subtree of a DOM. In this section of the tutorial, you'll experiment with that option.

> **Note:**
> The code discussed in this section is in TransformationApp03.java. The output is in TransformationLog03.

The only difference in the process is that now you will create a DOMSource using a node in the DOM, rather than the entire DOM. The first step will be to import the classes you need to get the node you want. Add the code highlighted below to do that:

```
import org.w3c.dom.DOMException;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
```

The next step is to find a good node for the experiment. Add the code highlighted below to select the first `<slide>` element:

```
   try {
      File f = new File(argv[0]);
      DocumentBuilder builder = factory.newDocumentBuilder();
      document = builder.parse(f);

      // Get the first <slide> element in the DOM
      NodeList list = document.getElementsByTagName("slide");
      Node node = list.item(0);
```

Finally, make the changes shown below to construct a source object that consists of the subtree rooted at that node:

```
      ~~DOMSource source = new DOMSource(document);~~
      DOMSource source = new DOMSource(node);
      StreamResult result = new StreamResult(System.out);
      transformer.transform(source, result);
```

Now run the app. Your output should look like this:

```
 <?xml version="1.0" encoding="UTF-8"?>
<slide type="all">
      <title>Wake up to WonderWidgets!</title>
    </slide>
```

Clean Up

Because it will be easiest to do now, make the changes shown below to back out the additions you made in this section. (TransformationApp04.java contains these changes.)

```
Import org.w3c.dom.DOMException;
~~import org.w3c.dom.Node;~~
~~import org.w3c.dom.NodeList;~~
...
   try {
      ...
      ~~// Get the first <slide> element in the DOM~~
      ~~NodeList list = document.getElementsByTagName("slide");~~
      ~~Node node = list.item(0);~~
```

```
...
DOMSource source = new DOMSource(node);
StreamResult result = new StreamResult(System.out);
transformer.transform(source, result);
```

Summary

At this point, you've seen how to use a transformer to write out a DOM, and how to use a subtree of a DOM as the source object in a transformation. In the next section, you'll see how to use a transformer to create XML from any data structure you are capable of parsing.

Top Contents Index Glossary

1. Introducing XSLT and XPath

The XML Stylesheet Language (XSL) has three major subcomponents:

- **XSL-FO**
  The "flow object" standard. By far the largest subcomponent, this standard gives mechanisms for describing font sizes, page layouts, and how information "flows" from one page to another. This subcomponent is *not* covered by JAXP, nor is it included in this tutorial.

- **XSLT**
  This the transformation language, which lets you transform XML into some other format. For example, you might use XSLT to produce HTML, or a different XML structure. You could even use it to produce plain text or to put the information in some other document format. (And as you'll see in Generating XML from an Arbitrary Data Structure, a clever application can press it into service to manipulate non-XML data, as well.)

- **XPath**
  At bottom, XSLT is a language that lets you specify what sorts of things to do when a particular element is encountered. But to write a program for different parts of an XML data structure, you need to be able to specify the part of the structure you are talking about at any given time. XPath is that specification language. It is an addressing mechanism that lets you specify a path to an element so, for example, <article><title> can be distinguished from <person><title>. That way, you can describe different kinds of translations for the different <title> elements.

The remainder of this section describes the XSLT package structure, and discusses the XPath addressing mechanism in a bit more depth.

Link Summary

Local Links

- Generating XML from an Arbitrary Data Structure
- Transforming XML Data with XSLT

External Links

- XPath Specification

Glossary Terms

DOM, SAX, URI, XSL

The XSLT Packages

There XSLT packages break down as follows:

javax.xml.transform
: This package defines the factory class you use to get a `Transformer` object. You then configure the transformer with input (Source) and output (Result) objects, and invoke its `transform()` method to make the transformation happen. The source and result objects are created using classes from one of the other three packages.

javax.xml.transform.dom
: Defines the `DOMSource` and `DOMResult` classes that let you use a DOM as an input to or output from a transformation.

javax.xml.transform.sax
: Defines the `SAXSource` and `SAXResult` classes that let you use a SAX event generator as input to a transformation, or deliver SAX events as output to a SAX event processor.

javax.xml.transform.stream
: Defines the `StreamSource` and `StreamResult` classes that let you use an I/O stream as an input to or output from a transformation.

How XPath Works

The XPath specification is the foundation for a variety of specifications, including XSLT and linking/addressing specifications like XPointer. So an understanding of XPath is fundamental to a lot of advanced XML usage. This section provides a thorough introduction to XSLT, so you can refer to as needed later on.

> **Note:**
> In this tutorial, you won't actually use XPath until you get to the last page of this section, Transforming XML Data with XSLT. So, if you like, you can skip this section and go on ahead to the next page, Writing Out a DOM as an XML File. (When you get to the last page, there will be a note that refers you back here, so you don't forget!)

In general, an XPath expression specifies a *pattern* that selects a set of XML nodes. XSLT templates then use those patterns when applying transformations. (XPointer, on the other hand, adds mechanisms for defining a *point* or a *range*, so that XPath expressions can be used for addressing.)

The nodes in an XPath expression refer to more than just elements. They also refer to text and attributes, among other things. In fact, the XPath specification defines an abstract document model that defines seven different kinds of nodes:

- root
- element
- text
- attribute
- comment
- processing instruction
- namespace

> **Note:**
> The root element of the XML data is modeled by an *element* node. The XPath root node contains the document's root element, as well as other information relating to the document.

The data model is described in the last section of the XPath Specification, Section 5. (Like many specifications, it is frequently helpful to start reading near the end! Frequently, many of the important terms and underlying assumptions are documented there. That sequence has often been the "magic key" that unlocks the contents of a W3C specification.)

In this abstract model, syntactic distinctions disappear, and you are left with a normalized view of the data. In a text node, for example, it makes no difference whether the text was defined in a CDATA section, or if it included entity references;. The text node will consist of normalized data, as it exists after all parsing is complete. So the text will contain a "<" character, regardless of whether an entity reference like `&lt;` or a CDATA section was used to include it. (Similarly for the "&" character.)

In this section of the tutorial, we'll deal mostly with element nodes and text nodes. For the other addressing mechanisms, see the XPath Specification.

Basic XPath Addressing

An XML document is a tree-structured (hierarchical) collection of nodes. Like a hierarchical directory structure, it is useful to specify a *path* that points a particular node in the hierarchy. (Hence the name of the specification: XPath). In fact, much of the notation of directory paths is carried over intact:

- The forward slash (/) is used as a path separator.
- An absolute path from the root of the document starts with a /.
- A relative path from a given location starts with anything else.
- A double period (..) indicates the parent of the current node.
- A single period . indicates the current node.

In an xHTML document, for example, the path `/h1/h2/` would indicate an h2 element under an h1. (Recall that in XML, element names are case sensitive, so this kind of specification works much better in xHTML than it would in HTML.)

In a pattern-matching specification like XSLT, the specification `/h1/h2` selects *all* h2 elements that lie under an h1 element. To select a specific h2 element, square brackets ([]) are used for indexing (like those used for arrays). The path `/h1[4]/h2[5]` would therefore select the fifth h2 element under the fourth h1 element.

> **Note:**
> In xHTML, all element names are in lowercase. But as a matter of style, uppercase names are easier to read and easier to write about. (Although they are admittedly harder to write.) For the remainder of XPATH tutorial, then, and for the section on using XSLT transforms, all XML element names will be in uppercase. (Attribute names, on the other hand, will remain in lowercase.)

As you've seen, a name in XPath specification refers to an element. To refer to attribute, you prefix it's name with an "@" sign. For example, `@type` refers to the `type` attribute of an element. Assuming you have an XML document with `list` elements, for example, the expression `list/@type` selects the `type` attribute of the `list` element.

> **Note:**
> (Since the expression does not begin with /, the reference specifies a list node relative to the current context -- whatever position in the document that happens to be.)

Basic XPath Expressions

The full range of XPath expressions takes advantage of the wildcards, operators, and functions that XPath defines. You'll be learning more about those shortly. Here, we'll take a look at a couple of the most common XPath expressions, simply to introduce the concept.

The expression `@type="unordered"` specifies an attribute named type whose value is "unordered". So an expression like `LIST/@type` specifies the `type attribute` of a `LIST` element.

But now for something a little different! In XPath, the square-bracket notation (`[]`) normally associated with indexing is extended to specify selection-criteria. For example, the expression `LIST[@type="unordered"]` selects all `LIST` elements whose `type` value is "unordered".

Similar expressions exist for elements, where each element has an associated *string-value*. (You'll see how the string-value is determined for a complicated element in a little while. For now, we'll stick with super-simple elements that have a single text string.)

Suppose you model what's going on in your organization with an XML structure that consists of `PROJECT` elements and `ACTIVITY` elements that have a text string with the project name, multiple `PERSON` elements to list the people involved and, optionally, a `STATUS` element that records the projects status. Here are some more examples that use the extended square-bracket notation:

- `/PROJECT[.="MyProject"]` selects a `PROJECT` named "MyProject".
- `/PROJECT[STATUS]` -- selects all projects that have a `STATUS` child element.
- `/PROJECT[STATUS="Critical"]` -- selects all projects that have a STATUS child element with the string-value "Critical".

Combining Index Addresses

The XPath specification defines quite a few addressing mechanisms, and they can be combined in many different ways. As a result, XPath delivers a lot of expressive power for a relatively simple specification. This section illustrates two more interesting combinations:

- `LIST[@type="ordered"][3]` -- selects all LIST elements of type "ordered", and returns the third.
- `LIST[3][@type="ordered"]` -- selects the third LIST element, but only if it is of "ordered" type.

  Note:
  Many more combinations of address operators are listed in section 2.5 of the XPath Specification. This is arguably the most useful section of the spec for defining an XSLT transform.

Wildcards

By definition, an unqualified XPath expression selects a set of XML nodes that matches that specified pattern. For example, `/HEAD` matches all top-level `HEAD` entries, while `/HEAD[1]` matches only the first. But XPath expressions can also contain one of several wildcards to broaden the scope of the pattern matching:

`*`	matches any element node (not attributes or text)
`node()`	matches all nodes of any kind: element nodes, text nodes, attribute nodes, processing instruction nodes, namespace nodes, and comment nodes.
`@*`	matches all attribute nodes

In the project database example, for instance, `/*/PERSON[.="Fred"]` matches any `PROJECT` or `ACTIVITY` element that includes Fred.

Extended-Path Addressing

So far, all of the patterns we've seen have specified an exact number of levels in the hierarchy. For example, `/HEAD` specifies any `HEAD` element at the first level in the hierarchy, while `/*/*` specifies any element at the second level in the hierarchy. To specify an indeterminate level in the hierarchy, use a double forward slash (`//`). For example, the XPath expression `//PARA` selects all `paragraph` elements in a document, wherever they may be found.

The `//` pattern can also be used within a path. So the expression `/HEAD/LIST//PARA` indicates all paragraph elements in a subtree that begins from `/HEAD/LIST`.

XPath Data Types and Operators

XPath expressions yield either a set of nodes,: a string, a boolean (true/false value), or a number. Expressions can also be created using one of several operations on these values:

|	Alternative. So `PARA|LIST` selects all `PARA` and `LIST` elements.
or, and	Returns the or/and of two boolean values.
=, !=	Equal or not equal, for booleans, strings, and numbers.
<, >, <=, >=	Less than, greater than, less than or equal to, greater than or equal to -- for numbers.
+, -, *, div, mod	Add, subtract, multiply, floating-point divide, and modulus (remainder) operations (e.g. 6 mod 4 = 2)

Finally, expressions can be grouped in parentheses, so you don't have to worry about operator precedence. (Which, for those of you who who are good at such things, is roughly the same as that shown in the table.)

String-Value of an Element

Before going on, it's worthwhile to understand how the string-value of more complex element is determined. We'll do that now.

The string-value of an element is the concatenation of all descendent text nodes, no matter how deep. So, for a "mixed-model" XML data element like this:

```
<PARA>This paragraph contains a <B>bold</B> word</PARA>
```

the string-value of `<PARA>` is "This paragraph contains a bold word". In particular, note that `<B>` is a child of `<PARA>` and that the text contained in all children is concatenated to form the string-value.

Also, it is worth understanding that the text in the abstract data model defined by XPath is fully normalized. So whether the XML structure contains the entity reference "`&lt;`" or "<" in a CDATA section, the element's string-value will contain the "<" character. Therefore, when generating HTML or XML with an XSLT stylesheet, occurrences of "<" will have to be converted to `&lt;` or enclosed in a CDATA section. Similarly, occurrence of "&" will need to be converted to `&amp;`.

XPath Functions

This section ends with an overview of the XPath functions. You can use XPath functions to select a collection of nodes in the same way that you would use an element-specification. Other functions return a string, a number, or a boolean value. For example, the expression `/PROJECT/text( )` gets the string-value of project nodes.

Many functions depend on the current context. In the example above, the *context* for each invocation of the `text()` function is the `PROJECT` node that is currently selected.

There are many XPath functions -- too many to describe in detail here. This section provides a quick listing that shows the available XPath functions, along with a summary of what they do.

> **Note:**
> Skim the list of functions to get an idea of what's there. For more information, see Section 4 of the XPath Specification.

Node-set functions

Many XPath expressions select a set of nodes. In essence, they return a *node-set*. One function does that, too.

- **`id(...)`** -- returns the node with the specified id.
  (Elements only have an ID when the document has a DTD, which specifies which attribute has the `ID` type.)

Positional functions

These functions return positionally-based numeric values.

- **`last()`** -- returns the index of the last element. Ex: `/HEAD[last()]` selects the last `HEAD` element.
- **`position()`** -- returns the index position. Ex: `/HEAD[position() <= 5]` selects the first five `HEAD` elements
- **`count(...)`** -- returns the count of elements. Ex: `/HEAD[count(HEAD)=0]` selects all `HEAD` elements that have no subheads.

String functions

These functions operate on or return strings.

- **`concat(string, string, ...)`** -- concatenates the string values
- **`starts-with(string1, string2)`** -- returns true if string1 starts with string2
- **`contains(string1, string2)`** -- returns true if string1 contains string2
- **`substring-before(string1, string2)`** -- returns the start of string1 before string2 occurs in it
- **`substring-after(string1, string2)`** -- returns the remainder of string1 after string2 occurs in it
- **`substring(string, idx)`** -- returns the substring from the index position to the end, where the index of the first char = 1
- **`substring(string, idx, len)`** -- returns the substring from the index position, of the specified length
- **`string-length()`** -- returns the size of the context-node's string-value
- **`string-length(string)`** -- returns the size of the specified string
- **`normalize-space()`** -- returns the normalized string-value of the current node (no leading or trailing whitespace, and sequences of whitespace characters converted to a single space)
- **`normalize-space(string)`** -- returns the normalized string-value of the specified string
- **`translate(string1, string2, string3)`** -- converts string1, replacing occurences of characters in string2 with the corresponding character from string3

  Note:
  XPath defines 3 ways to get the text of an element: text(), string(object), and the string-value implied by an element name in an expression like this: `/PROJECT[PERSON="Fred"]`.

Boolean functions

These functions operate on or return boolean values..

- **`not(...)`** -- negates the specified boolean value
- **`true()`** -- returns true
- **`false()`** -- returns false

- **`lang(string)`** -- returns true if the language of the context node (specified by `xml:Lang` attributes) is the same as (or a sublanguage of) the specified language. Ex: `Lang("en")` is true for `<PARA xml:Lang="en">...</PARA>`

Numeric functions

These functions operate on or return numeric values.

- **`sum(...)`** -- returns the sum of the numeric value of each node in the specified node-set
- **`floor(N)`** -- returns the largest integer that is not greater than N
- **`ceiling(N)`** -- returns the smallest integer that is greater than N
- **`round(N)`** -- returns the integer that is closest to N

Conversion functions

These functions convert one data type to another.

- **`string(...)`** -- returns the string value of a number, boolean, or node-set
- **`boolean(...)`** -- returns the boolean-equivalent for a number, string, or node-set
  (a non-zero number, a non-empty node-set, and a non-empty string are all true)
- **`number(...)`** -- returns the numeric value of a boolean, string, or node-set
  (true is 1, false is 0, a string containing a number becomes that number, the string-value of a node-set is converted to a number)

Namespace functions

These functions let you determine the namespace-characteristics of a node.

- **`local-name()`** -- returns the name of the current node, minus the namespace-extension
- **`local-name(...)`** -- returns the name of the first node in the specified node set, minus the namespace-extension
- **`namespace-uri()`** -- returns the namespace URI from the current node
- **`namespace-uri(...)`** -- returns the namespace URI from the first node in the specified node set
- **`name()`** -- returns the expanded name (URI + local name) of the current node
- **`name(...)`** -- returns the expanded name (URI + local name) of the first node in the specified node set

Summary

XPath operators, functions, wildcards, and node-addressing mechanisms can be combined in wide variety of ways. The introduction you've had so far should give you a good head start at specifying the pattern you need for any particular purpose.

1. Reading XML Data into a DOM

In this section of the tutorial, you'll construct a Document Object Model (DOM) by reading in an existing XML file. In the following sections, you'll see how to display the XML in a Swing tree component and practice manipulating the DOM.

> **Note:**
> In the next part of the tutorial, Using XSLT, you'll see how to write out a DOM as an XML file. (You'll also see how to convert an existing data file into XML with relative ease.)

Link Summary

Local Links

- Creating and Manipulating a DOM
- Compiling the Program
- Running the Program
- Handling Errors
- Using the Validating Parser
- Using Namespaces
- Using XSLT

Exercise Links

- DomEcho01.java
- slideSample01.xml
- slideSample01-xml.html

API Links

- DocumentBuilder
- Document

External Links

- Level 1 DOM specification

Glossary Terms

DOM, namespace, SAX, URI, validating parser

Creating the Program

The Document Object Model (DOM) provides APIs that let you create nodes, modify them, delete and rearrange them. So it is relatively easy to create a DOM, as you'll see in later in section 5 of this tutorial, Creating and Manipulating a DOM.

Before you try to create a DOM, however, it is helpful to understand how a DOM is structured. This series of exercises will make DOM internals visible by displaying them in a Swing JTree.

Create the Skeleton

Now that you've had a quick overview of how to create a DOM, let's build a simple program to read an XML document into a DOM then write it back out again.

> **Note:**
> The code discussed in this section is in DomEcho01.java. The file it operates on is slideSample01.xml. (The browsable version is slideSample01-xml.html.)

Start with a normal basic logic for an app, and check to make sure that an argument has been supplied on the command line:

```
public class DomEcho {

    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: java DomEcho filename");
            System.exit(1);
        }
    }// main

}// DomEcho
```

Import the Required Classes

In this section, you're going to see all the classes individually named. That's so you can see where each class comes from when you want to reference the API documentation. In your own apps, you may well want to replace import statements like those below with the shorter form: `javax.xml.parsers.*`.

Add these lines to import the JAXP APIs you'll be using:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
```

Add these lines for the exceptions that can be thrown when the XML document is parsed:

```
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
```

Add these lines to read the sample XML file and identify errors:

```
import java.io.File;
import java.io.IOException;
```

Finally, import the W3C definition for a DOM and DOM exceptions:

```
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
```

Note:
DOMExceptions are only thrown when traversing or manipulating a DOM. Errors that occur during parsing are reporting using a different mechanism that is covered below.

Declare the DOM

The org.w3c.dom.Document class is the W3C name for a Document Object Model (DOM). Whether you parse an

XML document or create one, a Document instance will result. We'll want to reference that object from another method later on in the tutorial, so define it as a global object here:

```
public class DomEcho
{
    static Document document;

    public static void main(String argv[])
    {
```

It needs to be `static`, because you're going to you generate it's contents from the `main` method in a few minutes.

Handle Errors

Next, put in the error handling logic. This code is very similar to the logic you saw in Handling Errors in the SAX tutorial, so we won't go into it in detail here. The major point worth noting is that a JAXP-conformant document builder is required to report SAX exceptions when it has trouble parsing the XML document. The DOM parser does not have to actually use a SAX parser internally, but since the SAX standard was already there, it seemed to make sense to use it for reporting errors. As a result, the error-handling code for DOM and SAX applications are very similar:

```
public static void main(String argv[])
{
    if (argv.length != 1) {
        ...
    }

    try {

    } catch (SAXException sxe) {
       // Error generated during parsing)
       Exception  x = sxe;
       if (sxe.getException() != null)
           x = sxe.getException();
       x.printStackTrace();

    } catch (ParserConfigurationException pce) {
       // Parser with specified options can't be built
       pce.printStackTrace();

    } catch (IOException ioe) {
       // I/O error
       ioe.printStackTrace();
    }
}// main
```

The major difference between this code and the SAX error-handling code is that the DOM parser does not throw SAXParseException's, but only SAXException's.

Instantiate the Factory

Next, add the code highlighted below to obtain an instance of a factory that can give us a document builder:

```
public static void main(String argv[])
{
    if (argv.length != 1) {
        ...
    }
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    try {
```

Get a Parser and Parse the File

Now, add the code highlighted below to get a instance of a builder, and use it to parse the specified file:

```
try {
   DocumentBuilder builder = factory.newDocumentBuilder();
   document = builder.parse( new File(argv[0]) );
} catch (SAXParseException spe) {
```

Save This File!
By now, you should be getting the idea that every JAXP application starts pretty much the same way. You're right! Save this version of the file as a template. You'll use it later on as the basis for XSLT transformation app.

Run the Program

Throughout most of the DOM tutorial, you'll be using the sample slideshows you created in the SAX section. In particular, you'll use `slideSample01.xml`, a simple XML file with nothing much in it, and `slideSample10.xml`, a more complex example that includes a DTD, processing instructions, entity references, and a CDATA section.

For instructions on how to compile and run your program, see Compiling the Program and Running the Program, from the SAX tutorial. Substitute "DomEcho" for "Echo" as the name of the program, and you're ready to roll.

For now, just run the program on slideSample01.xml. If it ran without error, you have successfully parsed an XML document and constructed a DOM. Congratulations!

Note:
You'll have to take my word for it, for the moment, because at this point you don't have any way to display the results. But that is feature is coming shortly...

Additional Information

Now that you have successfully read in a DOM, there are one or two more things you need to know in order to use DocumentBuilder effectively. Namely, you need to know about:

- Configuring the Factory
- Handling Validation Errors

Configuring the Factory

By default, the factory returns a nonvalidating parser that knows nothing about namespaces. To get a validating parser, and/or one that understands namespaces, you configure the factory to set either or both of those options using the command(s) highlighted below:

```
public static void main(String argv[])
{
    if (argv.length != 1) {
        ...
    }
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setValidating(true);
    factory.setNamespaceAware(true);
    try {
        ...
```

> **Note:**
> JAXP-conformant parsers are not required to support all combinations of those options, even though the reference parser does. If you specify an invalid combination of options, the factory generates a ParserConfigurationException when you attempt to obtain a parser instance.

You'll be learning more about how to use namespaces in the last section of the DOM tutorial, Using Namespaces. To complete this section, though, you'll want to learn something about...

Handling Validation Errors

Remember when you were wading through the SAX tutorial, and all you really wanted to do was construct a DOM? Well, here's when that information begins to pay off.

Recall that the default response to a validation error, as dictated by the SAX standard, is to do nothing. The JAXP standard requires throwing SAX exceptions, so you exactly the same error handling mechanisms as you used for a SAX app. In particular, you need to use the DocumentBuilder's `setErrorHandler` method to supply it with an object that implements the SAX ErrorHandler interface.

> Note:
> DocumentBuilder also has a `setEntityResolver` method you can use

The code below uses an anonymous inner class adapter to provide that ErrorHandler. The highlighted code is the part that makes sure validation errors generate an exception.

```
    builder.setErrorHandler(
      new org.xml.sax.ErrorHandler() {
          // ignore fatal errors (an exception is guaranteed)
          public void fatalError(SAXParseException exception)
          throws SAXException {
          }

          // treat validation errors as fatal
          public void error(SAXParseException e)
          throws SAXParseException
          {
            throw e;
          }

          // dump warnings too
          public void warning(SAXParseException err)
          throws SAXParseException
          {
            System.out.println("** Warning"
               + ", line " + err.getLineNumber()
               + ", uri " + err.getSystemId());
            System.out.println("   " + err.getMessage());
          }
      }
    );
```

This code uses an anonymous inner class to generate an instance of an object that implements the ErrorHandler interface. Since it has no class name, it's "anonymous". You can think of it as an "ErrorHandler" instance, although technically it's a no-name instance that implements the specified interface. The code is substantially the same as that described the Handling Errors section of the SAX tutorial. For a more background on validation issues, refer to Using the Validating Parser in that part of the tutorial.

Looking Ahead

In the next section, you'll display the DOM structure in a JTree and begin explore its structure. For example, you'll see how entity references and CDATA sections appear in the DOM. And perhaps most importantly, you'll see how text nodes (which contain the actual data) reside *under* element nodes in a DOM.

2a. Displaying a DOM Hierarchy

To create a Document Object Hierarchy (DOM) or manipulate one, it helps to have a clear idea of how nodes in a DOM are structured. In this section of the tutorial, you'll expose the internal structure of a DOM.

Link Summary
Exercise Links   • DomEcho02.java   ***External Links***   • DOM 2 Core Specification   • Understanding the TreeModel

Echoing Tree Nodes

What you need at this point is a way to expose the nodes in a DOM so can see what it contains. To do that, you'll convert a DOM into a JTreeModel and display the full DOM in a JTree. It's going to take a bit of work, but the end result will be a diagnostic tool you can use in the future, as well as something you can use to learn about DOM structure now.

Convert DomEcho to a GUI App

Since the DOM is a tree, and the Swing JTree component is all about displaying trees, it makes sense to stuff the DOM into a JTree, so you can look it. The first step in that process is to hack up the DomEcho program so it becomes a GUI application.

> **Note:**
> The code discussed in this section is in `DomEcho02.java`.

Add Import Statements

Start by importing the GUI components you're going to need to set up the application and display a JTree:

```
// GUI components and layouts
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
```

Later on in the DOM tutorial, we'll going to tailor the DOM display to generate a user-friendly version of the JTree display. When the user selects an element in that tree, you'll be displaying subelements in an adjacent editor pane. So, while we're doing the setup work here, import the components you need to set up a divided view (JSplitPane) and to display the text of the subelements (JEditorPane):

```
import javax.swing.JSplitPane;
import javax.swing.JEditorPane;
```

Add a few support classes you're going to need to get this thing off the ground:

```
// GUI support classes
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.WindowEvent;
import java.awt.event.WindowAdapter;
```

Finally, import some classes to make a fancy border:

```
// For creating borders
import javax.swing.border.EmptyBorder;
import javax.swing.border.BevelBorder;
import javax.swing.border.CompoundBorder;
```

(These are optional. You can skip them and the code that depends on them if you want to simplify things.)

Create the GUI Framework

The next step is to convert the app into a GUI application. To do that, the static main method will create an instance of the main class, which will have become a GUI pane.

Start by converting the class into a GUI pane by extending the Swing JPanel class:

```
public class DomEcho02 extends JPanel
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;
    ...
```

While you're there, define a few constants you'll use to control window sizes:

```
public class DomEcho02 extends JPanel
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

    static final int windowHeight = 460;
    static final int leftWidth = 300;
    static final int rightWidth = 340;
    static final int windowWidth = leftWidth + rightWidth;
```

Now, in the main method, invoke a method that will create the outer frame that the GUI pane will sit in:

```
 public static void main(String argv[])
 {
    ...
    DocumentBuilderFactory factory ...
    try {
       DocumentBuilder builder = factory.newDocumentBuilder();
       document = builder.parse( new File(argv[0]) );
```

```
            makeFrame();

         } catch (SAXParseException spe) {
            ...
```

Next, you'll need to define the `makeFrame` method itself. It contains the standard code to create a frame, handle the exit condition gracefully, give it an instance of the main panel, size it, locate it on the screen, and make it visible:

```
       ...
     } // main

     public static void makeFrame()
     {
         // Set up a GUI framework
         JFrame frame = new JFrame("DOM Echo");
         frame.addWindowListener(new WindowAdapter() {
           public void windowClosing(WindowEvent e) {System.exit(0);}
         });

         // Set up the tree, the views, and display it all
         final DomEcho02 echoPanel = new DomEcho02();
         frame.getContentPane().add("Center", echoPanel );
         frame.pack();
         Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
         int w = windowWidth + 10;
         int h = windowHeight + 10;
         frame.setLocation(screenSize.width/3 - w/2, screenSize.height/2 - h/2);
         frame.setSize(w, h);
         frame.setVisible(true);
     } // makeFrame
```

Add the Display Components

The only thing left in the effort to convert the program to a GUI app is create the class constructor and make it create the panel's contents. Here is the constructor:

```
     public class DomEcho02 extends JPanel
     {
         ...
         static final int windowWidth = leftWidth + rightWidth;

         public DomEcho02()
         {
         } // Constructor
```

Here, you make use of the border classes you imported earlier to make a regal border (optional):

```
     public DomEcho02()
     {
        // Make a nice border
        EmptyBorder eb = new EmptyBorder(5,5,5,5);
        BevelBorder bb = new BevelBorder(BevelBorder.LOWERED);
```

```
         CompoundBorder cb = new CompoundBorder(eb,bb);
         this.setBorder(new CompoundBorder(CB,eb));

      } // Constructor
```

Next, create an empty tree and put it a JScrollPane so users can see its contents as it gets large:

```
      public DomEcho02()
      {
         ...

         // Set up the tree
         JTree tree = new JTree();

         // Build left-side view
         JScrollPane treeView = new JScrollPane(tree);
         treeView.setPreferredSize(
            new Dimension( leftWidth, windowHeight ));

      } // Constructor
```

Now create a non-editable JEditPane that will eventually hold the contents pointed to by selected JTree nodes:

```
      public DomEcho02()
      {
         ....

         // Build right-side view
         JEditorPane htmlPane = new JEditorPane("text/html","");
         htmlPane.setEditable(false);
         JScrollPane htmlView = new JScrollPane(htmlPane);
         htmlView.setPreferredSize(
             new Dimension( rightWidth, windowHeight ));

      }  // Constructor
```

With the left-side JTree and the right-side JEditorPane constructed, create a JSplitPane to hold them:

```
      public DomEcho02()
      {
         ....

         // Build split-pane view
         JSplitPane splitPane = new JSplitPane( JSplitPane.HORIZONTAL_SPLIT,
                                               treeView,
                                               htmlView );
         splitPane.setContinuousLayout( true );
         splitPane.setDividerLocation( leftWidth );
         splitPane.setPreferredSize(
             new Dimension( windowWidth + 10, windowHeight+10 ));

      }  // Constructor
```

With this code, you set up the JSplitPane so with a vertical divider. That produces a "horizontal split" between the tree and the editor pane. (More of a horizontal layout, really.) You also set the location of the divider so that the tree got the width it prefers, with the remainder of the window width allocated to the editor pane.

Finally, specify the layout for the panel and add the split pane:

```
public DomEcho02()
{
   ...

   // Add GUI components
   this.setLayout(new BorderLayout());
   this.add("Center", splitPane );

} // Constructor
```

Congratulations! The program is now a GUI app. You can run it now to see what the general layout will look like on screen. For reference, here is the completed constructor:

```
    public DomEcho02()
    {
       // Make a nice border
       EmptyBorder eb = new EmptyBorder(5,5,5,5);
       BevelBorder bb = new BevelBorder(BevelBorder.LOWERED);
       CompoundBorder CB = new CompoundBorder(eb,bb);
       this.setBorder(new CompoundBorder(CB,eb));

       // Set up the tree
       JTree tree = new JTree();

       // Build left-side view
       JScrollPane treeView = new JScrollPane(tree);
       treeView.setPreferredSize(
           new Dimension( leftWidth, windowHeight ));

       // Build right-side view
       JEditorPane htmlPane = new JEditorPane("text/html","");
       htmlPane.setEditable(false);
       JScrollPane htmlView = new JScrollPane(htmlPane);
       htmlView.setPreferredSize(
           new Dimension( rightWidth, windowHeight ));

       // Build split-pane view
       JSplitPane splitPane = new JSplitPane( JSplitPane.HORIZONTAL_SPLIT,
                                              treeView,
                                              htmlView );
       splitPane.setContinuousLayout( true );
       splitPane.setDividerLocation( leftWidth );
       splitPane.setPreferredSize(
            new Dimension( windowWidth + 10, windowHeight+10 ));
       // Add GUI components
       this.setLayout(new BorderLayout());
       this.add("Center", splitPane );
    } // Constructor
```

Create Adapters to Display the DOM in a JTree

Now that you have a GUI framework to display a JTree in, the next step is get the JTree to display the DOM. But a JTree wants to display a TreeModel. A DOM is a tree, but it's not a TreeModel. So you'll need to create an adapter class that makes the DOM look like a TreeModel to a JTree.

Now, when the TreeModel passes nodes to the JTree, JTree uses the `toString` function of those nodes to get the text to display in the tree. The standard `toString` function isn't going to be very pretty, so you'll need to wrap the DOM nodes in an AdapterNode that returns the text we want. What the TreeModel gives to the JTree, then, will in fact be AdapterNode objects that wrap DOM nodes.

> **Note:**
> The classes that follow are defined as inner classes. If you are coding for the 1.1 platform, you will need to define these class as external classes.

Define the AdapterNode Class

Start by importing the tree, event, and utility classes you're going to need to make this work:

```
    // For creating a TreeModel
    import javax.swing.tree.*;
    import javax.swing.event.*;
    Import java.util.*;

    public class DomEcho02 extends JPanel
    {
```

Moving back down to the end of the program, define a set of strings for the node element types:

```
           ...
        } // makeFrame

        // An array of names for DOM node-types
        // (Array indexes = nodeType() values.)
        static final String[] typeName = {
            "none",
            "Element",
            "Attr",
            "Text",
            "CDATA",
            "EntityRef",
            "Entity",
            "ProcInstr",
            "Comment",
            "Document",
            "DocType",
            "DocFragment",
            "Notation",
        };

    } // DomEcho
```

These are the strings that will be displayed in the JTree. The specification of these nodes types can be found in the Document Object Model (DOM) Level 2 Core Specification, under the specification for Node. That table is reproduced below, with the headings modified for clarity, and with the nodeType() column added:

Table of Node Type

Node	**nodeName()**	**nodeValue()**	**attributes**	**nodeType()**
Attr	name of attribute	value of attribute	null	2
CDATASection	#cdata-section	content of the CDATA Section	null	4
Comment	#comment	content of the comment	null	8
Document	#document	null	null	9
DocumentFragment	#document-fragment	null	null	11
DocumentType	document type name	null	null	10
Element	tag name	null	NamedNodeMap	1
Entity	entity name	null	null	6
EntityReference	name of entity referenced	null	null	5
Notation	notation name	null	null	12
ProcessingInstruction	target	entire content excluding the target	null	7
Text	#text	content of the text node	null	3

Suggestion:
Print this table and keep it handy. You need it when working with the DOM, because all of these types are intermixed in a DOM tree. So your code is forever asking, "Is this the kind of node I'm interested in?".

Next, define the AdapterNode wrapper for DOM nodes:

```
        static final String[] typeName = {
            ...
        };

        public class AdapterNode
        {
          org.w3c.dom.Node domNode;

          // Construct an Adapter node from a DOM node
          public AdapterNode(org.w3c.dom.Node node) {
            domNode = node;
          }

          // Return a string that identifies this node in the tree
          // *** Refer to table at top of org.w3c.dom.Node ***
          public String toString() {
            String s = typeName[domNode.getNodeType()];
            String nodeName = domNode.getNodeName();
            if (! nodeName.startsWith("#")) {
               s += ": " + nodeName;
            }
            if (domNode.getNodeValue() != null) {
               if (s.startsWith("ProcInstr"))
                  s += ", ";
               else
                  s += ": ";
               // Trim the value to get rid of NL's at the front
               String t = domNode.getNodeValue().trim();
               int x = t.indexOf("\n");
               if (x >= 0) t = t.substring(0, x);
               s += t;
            }
            return s;
          }

        } // AdapterNode

    } // DomEcho
```

This class declares a variable to hold the DOM node, and requires it to be specified as a constructor argument. It then defines the `toString` operation, which returns the node type from the String array, and then adds to that additional information from the node, to further identify it.

As you can see in the table of node types in org.w3c.dom.Node, every node has a type, and name, and a value, which may or may not be empty. In those cases where the node name starts with "#", that field duplicates the node type, so there is in point in including it. That explains the lines that read:

```
    if (! nodeName.startsWith("#")) {
        s += ": " + nodeName;
    }
```

The remainder of the `toString` method deserves a couple of notes, as well. For instance, these lines:

```
    if (s.startsWith("ProcInstr"))
        s += ", ";
    else
        s += ": ";
```

Merely provide a little "syntactic sugar". The type field for a Processing Instructions end with a colon (:) anyway, so those codes keep from doubling the colon.

The other interesting lines are:

```
String t = domNode.getNodeValue()Trim();
int x = t.indexOf("\n");
if (x >= 0) t = t.substring(0, x);
s += t;
```

Those lines trim the value field down to the first newline (linefeed) character in the field. If you leave those lines out, you will see some funny characters (square boxes, typically) in the JTree.

Note:
Recall that XML stipulates that all line endings are normalized to newlines, regardless of the system the data comes from. That makes programming quite a bit simpler.

Wrapping a DomNode and returning the desired string are the AdapterNode's major functions. But since the TreeModel adapter will need to answer questions like "How many children does this node have?" and satisfy commands like "Give me this node's Nth child", it will helpful to define a few additional utility methods. (The adapter could always access the DOM node and get that information for itself, but this way things are more encapsulated.)

Add the code highlighted below to return the index of a specified child, the child that corresponds to a given index, and the count of child nodes:

```
    public class AdapterNode
    {
      ...
      public String toString() {
        ...
      }

      public int index(AdapterNode child) {
        //System.err.println("Looking for index of " + child);
        int count = childCount();
        for (int i=0; i<count; i++) {
          AdapterNode n = this.child(i);
          if (child == n) return i;
        }
        return -1; // Should never get here.
      }

      Public AdapterNode child(int searchIndex) {
        //Note: JTree index is zero-based.
        org.w3c.dom.Node node = domNode.getChildNodes().item(searchIndex);
        return new AdapterNode(node);
      }

      public int childCount() {
          return domNode.getChildNodes().getLength();
      }
    } // AdapterNode

} // DomEcho
```

Note:

During development, it was only after I started writing the TreeModel adapter that I realized these were needed, and went back to add them. In just a moment, you'll see why.

Define the TreeModel Adapter

Now, at last, you are ready to write the TreeModel adapter. One of the really nice things about the JTree model is the relative ease with which you convert an existing tree for display. One of the reasons for that is the clear separation between the displayable view, which JTree uses, and the modifiable view, which the application uses. For more on that separation, see Understanding the TreeModel. For now, the important point is that to satisfy the TreeModel interface we only need to (a) provide methods to access and report on children and (b) register the appropriate JTree listener, so it knows to update its view when the underlying model changes.

Add the code highlighted below to create the TreeModel adapter and specify the child-processing methods:

```
  ...

} // AdapterNode

// This adapter converts the current Document (a DOM) into
// a JTree model.
Public class DomToTreeModelAdapter implements javax.swing.tree.TreeModel
{
  // Basic TreeModel operations
  public Object  getRoot() {
    //System.err.println("Returning root: " +document);
    return new AdapterNode(document);
  }
  public boolean isLeaf(Object aNode) {
    // Determines whether the icon shows up to the left.
    // Return true for any node with no children
    AdapterNode node = (AdapterNode) anode;
    if (node.childCount() > 0) return false;
    return true;
  }
  public int     getChildCount(Object parent) {
    AdapterNode node = (AdapterNode) parent;
    return node.childCount();
  }
  public Object  getChild(Object parent, int index) {
    AdapterNode node = (AdapterNode) parent;
    return node.child(index);
  }
  public int     getIndexOfChild(Object parent, Object child) {
    AdapterNode node = (AdapterNode) parent;
    return node.index((AdapterNode) child);
  }
  public void    valueForPathChanged(TreePath path, Object newValue) {
    // Null. We won't be making changes in the GUI
    // If we did, we would ensure the new value was really new
    // and then fire a TreeNodesChanged event.
  }
```

```
      } // DomToTreeModelAdapter

    } // DomEcho
```

In this code, the `getRoot` method returns the root node of the DOM, wrapped as an AdapterNode object. From here on, all nodes returned by the adapter will be AdapterNodes that wrap DOM nodes. By the same token, whenever the JTree asks for the child of a given parent, the number of children that parent has, etc., the JTree will be passing us an AdapterNode. We know that, because we control every node the JTree sees, starting with the root node.

JTree uses the isLeaf method to determine whether or not to display a clickable expand/contract icon to the left of the node, so that method returns true only if the node has children. In this method, we see the cast from the generic object JTree sends us to the AdapterNode object we know it has to be. *We* know it is sending us an adapter object, but the interface, to be general, defines objects, so we have to do the casts.

The next three methods return the number of children for a given node, the child that lives at a given index, and the index of a given child, respectively. That's all pretty straightforward.

The last method is invoked when the user changes a value stored in the JTree. In this app, we won't support that. But if we did, the app would have to make the change to the underlying model and then inform any listeners that a change had occurred. (The JTree might not be the only listener. In many an application it isn't, in fact.)

To inform listeners that a change occurred, you'll need the ability to register them. That brings us to the last two methods required to implement the TreeModel interface. Add the code highlighted below to define them:

```
public class DomToTreeModelAdapter ...
{
  ...
  public void    valueForPathChanged(TreePath path, Object newValue) {
    ...
  }

  private Vector listenerList = new Vector();
  public void addTreeModelListener( TreeModelListener listener ) {
    if ( listener != null && ! listenerList.contains( listener ) ) {
       listenerList.addElement( listener );
    }
  }
  public void removeTreeModelListener( TreeModelListener listener ) {
    if ( listener != null ) {
       listenerList.removeElement( listener );
    }
  }

} // DomToTreeModelAdapter
```

Since this app won't be making changes to the tree, these methods will go unused, for now. However, they'll be there in the future, when you need them.

Note:
This example uses Vector so it will work with 1.1 apps. If coding for 1.2 or later, though, I'd use the excellent collections framework instead:

```
private LinkedList listenerList = new LinkedList();
```

The operations on the List are then `add` and `remove`. To iterate over the list, as in the operations below, you would use:

```
Iterator it = listenerList.iterator();
while ( it.hasNext() ) {
  TreeModelListener listener = (TreeModelListener) it.next();
   ...
}
```

Here, too, are some optional methods you won't be using in this app. At this point, though, you have constructed a reasonable template for a TreeModel adapter. In the interests of completeness, you might want to add the code highlighted below. You can then invoke them whenever you need to notify JTree listeners of a change:

```
  public void removeTreeModelListener( TreeModelListener listener ) {
    ...
  }
  public void fireTreeNodesChanged( TreeModelEvent e ) {
    Enumeration listeners = listenerList.elements();
    while ( listeners.hasMoreElements() ) {
      TreeModelListener listener = (TreeModelListener) listeners.nextElement();
      listener.treeNodesChanged( e );
    }
  }
  public void fireTreeNodesInserted( TreeModelEvent e ) {
    Enumeration listeners = listenerList.elements();
    while ( listeners.hasMoreElements() ) {
       TreeModelListener listener = (TreeModelListener) listeners.nextElement();
       listener.treeNodesInserted( e );
    }
  }
  public void fireTreeNodesRemoved( TreeModelEvent e ) {
    Enumeration listeners = listenerList.elements();
    while ( listeners.hasMoreElements() ) {
      TreeModelListener listener = (TreeModelListener) listeners.nextElement();
      listener.treeNodesRemoved( e );
    }
  }
  public void fireTreeStructureChanged( TreeModelEvent e ) {
    Enumeration listeners = listenerList.elements();
    while ( listeners.hasMoreElements() ) {
      TreeModelListener listener = (TreeModelListener) listeners.nextElement();
      listener.treeStructureChanged( e );
    }
  }
} // DomToTreeModelAdapter
```

Note:
These methods are taken from the TreeModelSupport class described in Understanding the TreeModel. That

architecture was produced by Tom Santos and Steve Wilson, and is a lot more elegant than the quick hack going on here. It seemed worthwhile to put them here, though, so they would be immediately at hand when and if they're needed..

Finish it Up

At this point, you are basically done. All you need to do is jump back to the constructor and add the code to construct an adapter and deliver it to the JTree as the TreeModel:

```
// Set up the tree
JTree tree = new JTree(new DomToTreeModelAdapter());
```

You can now compile and run the code on an XML file. In the next section, you will do that, and explore the DOM structures that result.

2b. Examining the Structure of a DOM

In this section, you'll use the GUI-fied DomEcho app you created in the last section to visually examine a DOM. You'll see what nodes make up the DOM, and how they are arranged. With the understanding you acquire, you'll be well prepared to construct and modify Document Object Model structures in the future.

Link Summary

Local Links

- Table of Node Types
- Creating and Manipulating a DOM

Exercise Links

- DomEcho02.java
- slideSample01.xml
- slideSample01.xml
- slideSample10.xml
- slideSample10-xml.html
- slideshow3.dtd
- slideshow3-dtd.html
- copyright.xml
- copyright-xml.html
- xhtml.dtd
- xhtml-dtd.html

ternal Links

- DOM 2 Core Specification

ossary Terms

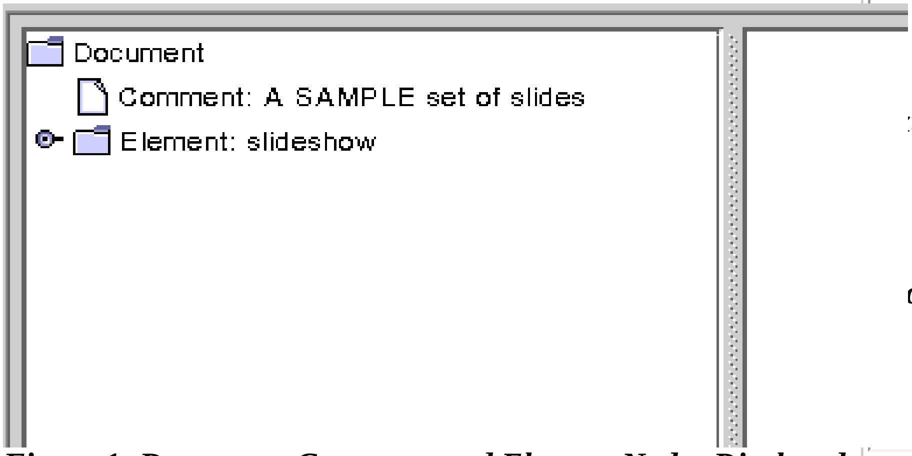
DTD

Displaying A Simple Tree

We'll start out by displaying a simple file, so you get an idea of basic DOM structure. Then we'll look at the structure that results when you include some of the more advanced XML elements.

> **Note:**
> The code used to create the figures in this section is in DomEcho02.java. The file displayed is slideSample01.xml. (The browsable version is slideSample01-xml.html.)

Figure 1 shows the tree you see when you run the DomEcho program on the first XML file you created in the DOM tutorial.



```
Document
    Comment: A SAMPLE set of slides
  Element: slideshow
```

Figure 1: Document, Comment, and Element Nodes Displayed

Recall that the first bit of text displayed for each node is the element `type`. After that comes the element `name`, if any, and then the element `value`. This view shows three element types: Document, Comment, and Element. There is only Document type for the whole tree -- that is the root node. The Comment node displays the `value` attribute, while the Element node displays the element `name`, "slideshow".

Compare the Table of Node Types with the code in the AdapterNode's `toString` method to see whether the name or value is being displayed for a particular node. If you need to make it more clear, modify the program to indicate which property is being displayed (for example, with N: *name*, V: *value*).

Expanding the slideshow element brings up the display shown in Figure 2.

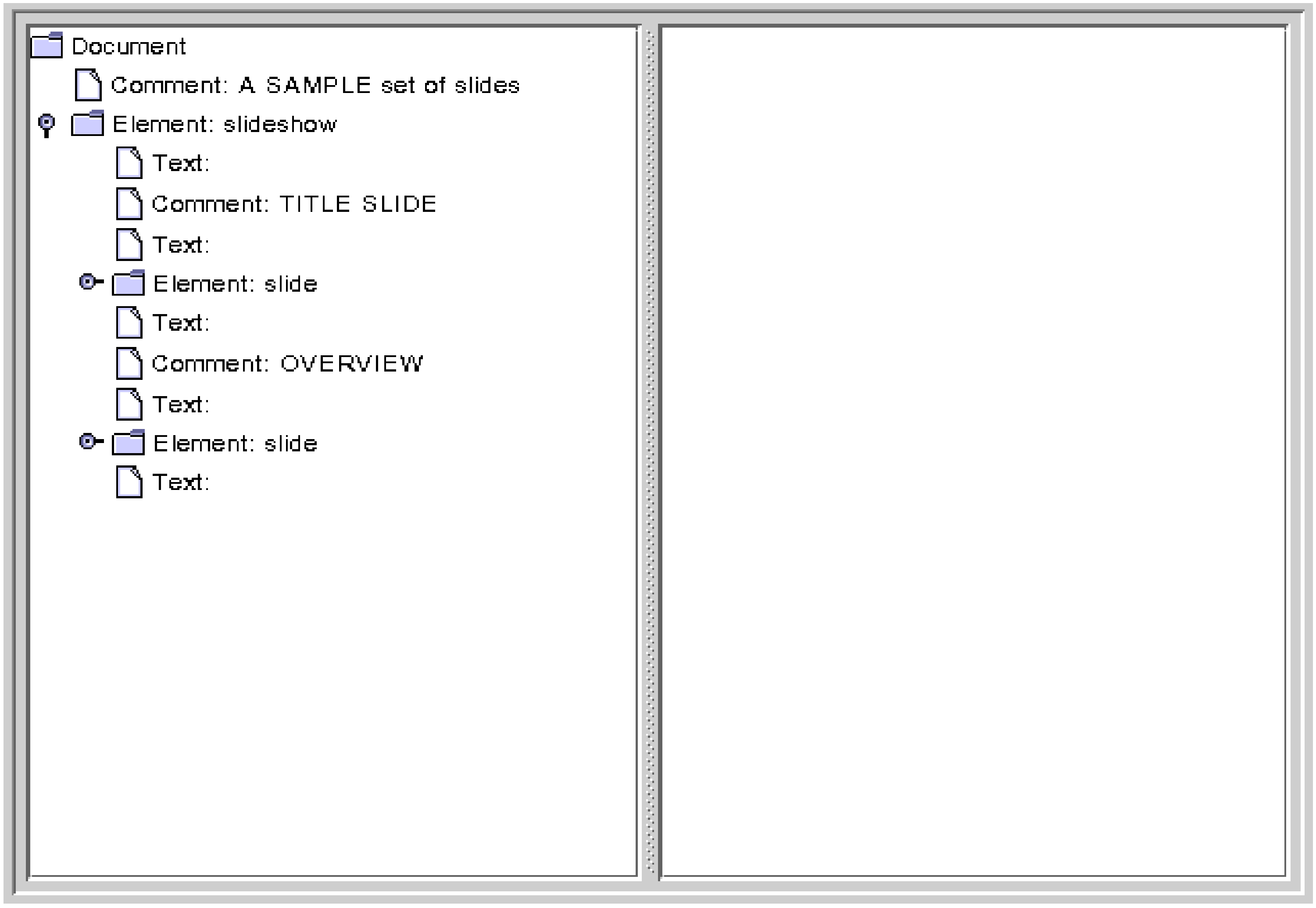

```
Document
    Comment: A SAMPLE set of slides
    Element: slideshow
        Text:
        Comment: TITLE SLIDE
        Text:
        Element: slide
        Text:
        Comment: OVERVIEW
        Text:
        Element: slide
        Text:
```

Figure 2: Element Node Expanded, No Attribute Nodes Showing

Here, you can see the Text nodes and Comment nodes that are interspersed between Slide elements. The empty Text nodes exist because there is no DTD to tell the parser that no text exists. (Generally, the vast majority of nodes in a DOM tree will be Element and Text nodes.)

Important!
Text nodes exist *under* element nodes in a DOM, and data is *always* stored in text nodes. Perhaps the most common error in DOM processing is to navigate to an element node and expect it to contain the data that is stored in the XML file. Not so! Even the simplest element node has a text node under it. For example, given `<size>12</size>`, there is an element node (`size`), *and a text node under it* which contains the actual data (12).

Notably absent from this picture are the Attribute nodes. An inspection of the table in org.w3c.dom.Node shows that there is indeed an Attribute node type. But they are not included as children in the DOM hierarchy. They are instead obtained via the Node interface `getAttributes` method.

> **Note:**
> The display of the text nodes is the reason for including the lines below in the AdapterNode's `toString` method. If your remove them, you'll see the funny characters (typically square blocks) that are generated by the newline characters that are in the text.
>
> ```
> String t = domNode.getNodeValue().trim();
> int x = t.indexOf("\n");
> if (x >= 0) t = t.substring(0, x);
> s += t;
> ```

Displaying a More Complex Tree

Here, you'll display the example XML file you created at the end of the SAX tutorial, to see how entity references, processing instructions, and CDATA sections appear in the DOM.

> **Note:**
> The file displayed in this section is slideSample10.xml. The slideSample10.xml file references slideshow3.dtd which, in turn, references copyright.xml and a (very simplistic) xhtml.dtd. (The browsable versions are slideSample10-xml.html, slideshow3-dtd.html, copyright-xml.html, and xhtml-dtd.html.)

Figure 3 shows the result of running the DomEcho app on slideSample10.xml, which includes a `DOCTYPE` entry that identifies the document's DTD.

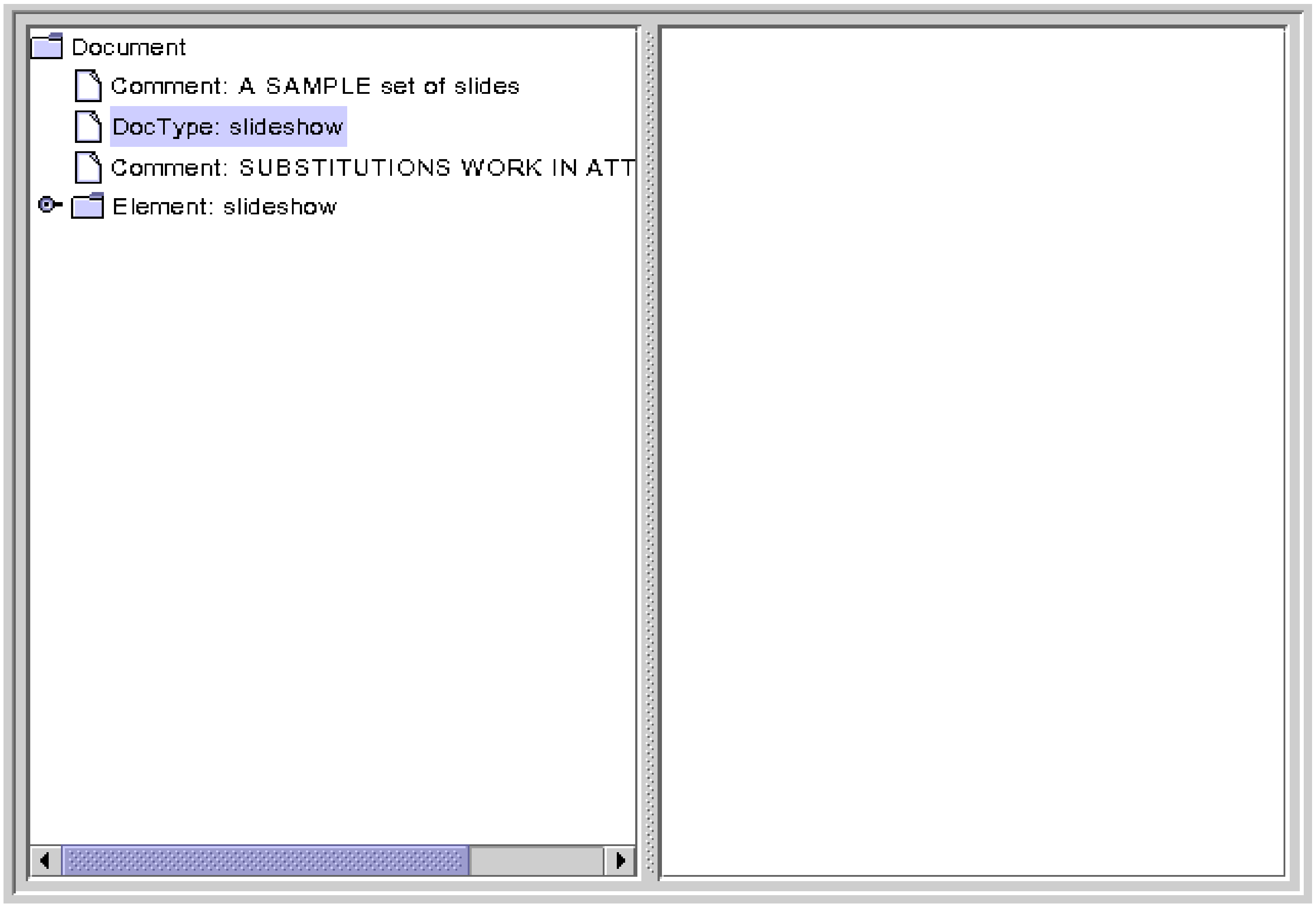
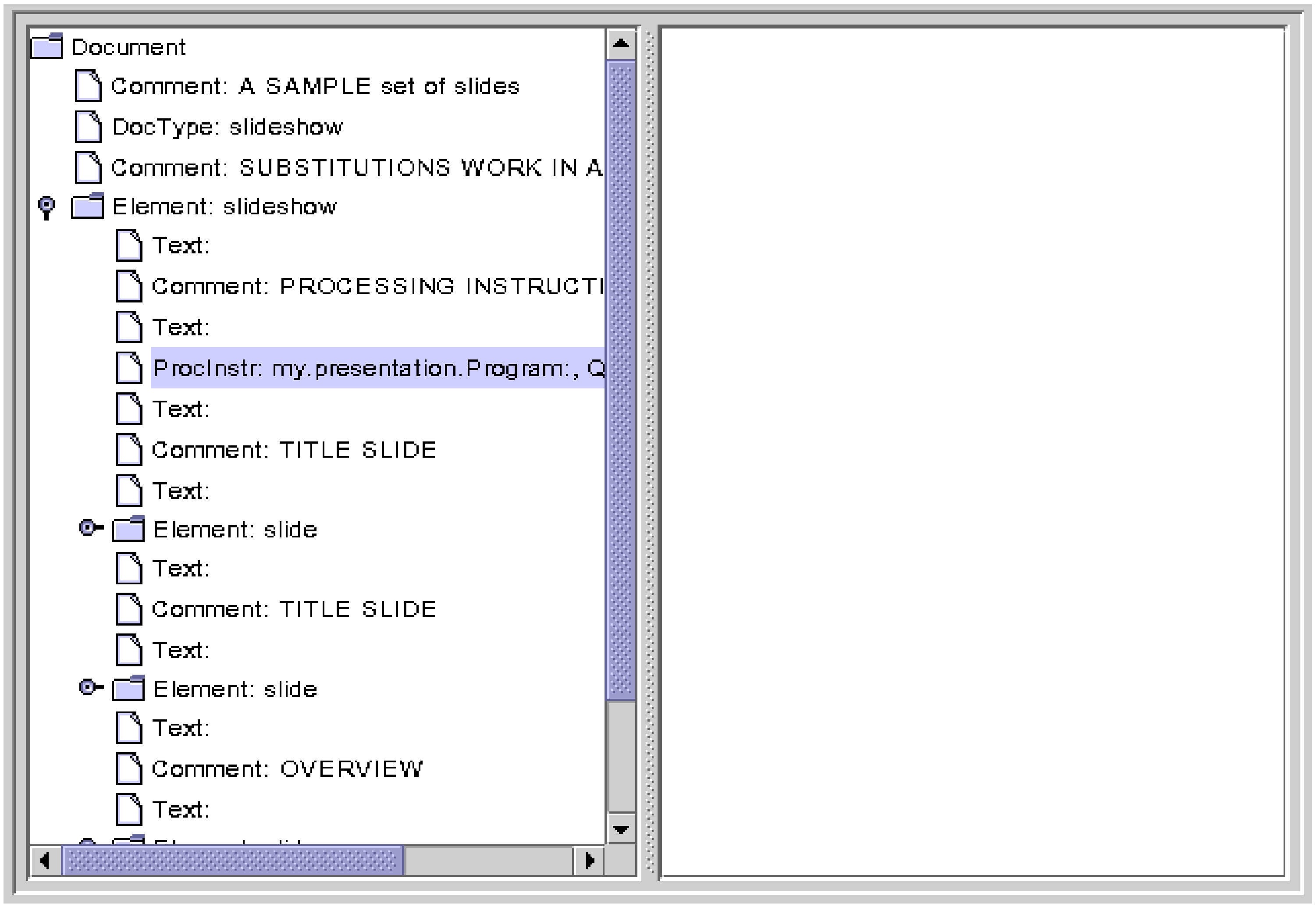

Document
Comment: A SAMPLE set of slides
DocType: slideshow
Comment: SUBSTITUTIONS WORK IN ATT
Element: slideshow

Figure 3: DocType Node Displayed

The DocType interface is actually an extension of w3c.org.dom.Node. It defines a `getEntities` method that you would use to to obtain Entity nodes -- the nodes that define entities like the `product` entity, which has the value "WonderWidgets". Like Attribute nodes, Entity nodes do not appear as children of DOM nodes.

When you expand the `slideshow` node, you get the display shown in Figure 4.



```
Document
    Comment: A SAMPLE set of slides
    DocType: slideshow
    Comment: SUBSTITUTIONS WORK IN A
    Element: slideshow
        Text:
        Comment: PROCESSING INSTRUCTI
        Text:
        ProcInstr: my.presentation.Program:, Q
        Text:
        Comment: TITLE SLIDE
        Text:
        Element: slide
        Text:
        Comment: TITLE SLIDE
        Text:
        Element: slide
        Text:
        Comment: OVERVIEW
        Text:
```

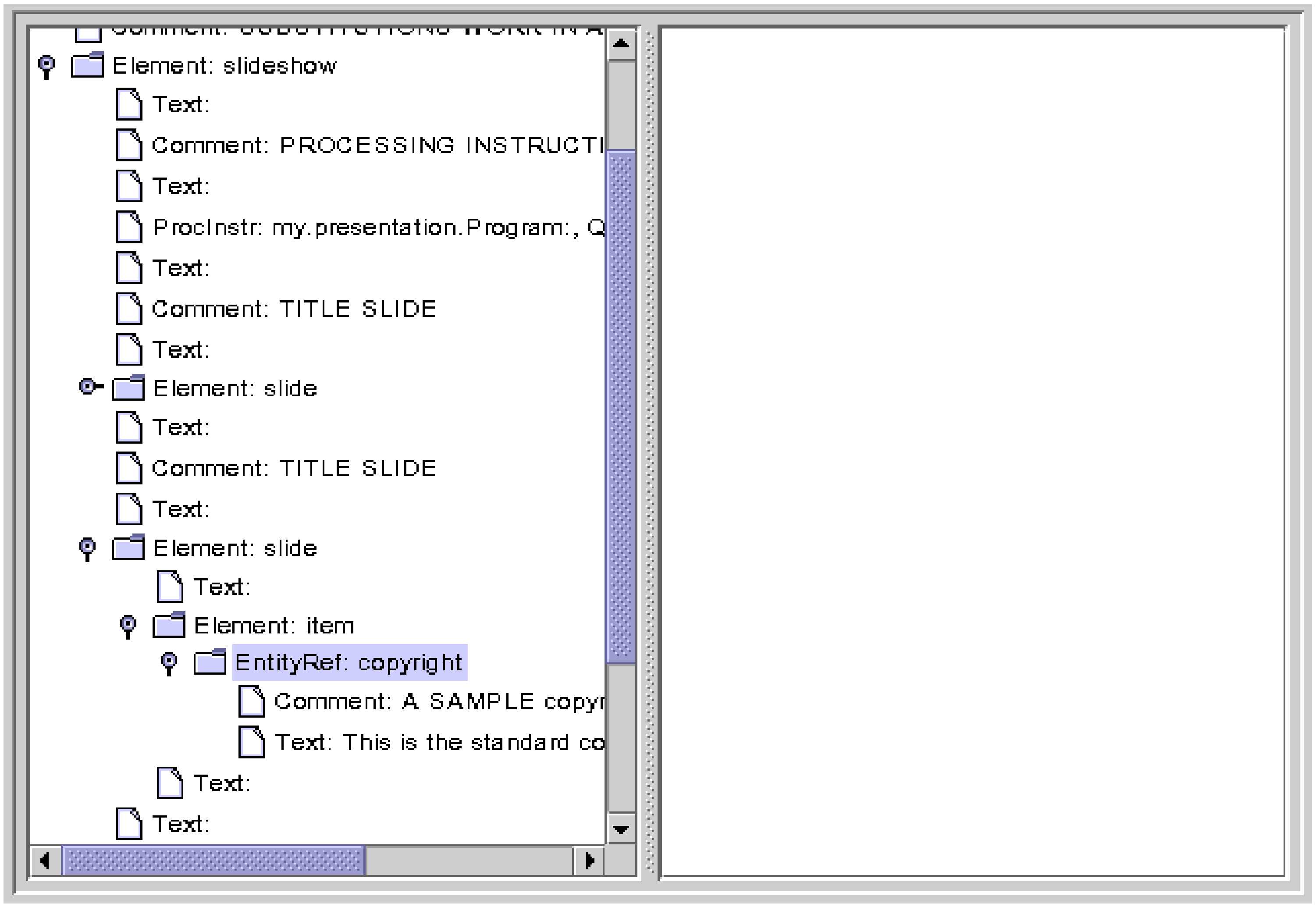
Figure 4: Processing Instruction Node Displayed

Here, the processing instruction node is highlighted, showing that those nodes do appear in the tree. The `name` property contains the target-specification, which identifies the app that the instruction is directed to. The `value` property contains the text of the instruction..

Note that empty text nodes are also shown here, even though the DTD specifies that a `slideshow` can contain `slide` elements only, never text. Logically, then, you might think that these nodes would not appear. (When this file was run through the SAX parser, those elements generated `ignorableWhitespace` events, rather than `character` events.)

The empty text elements are included because by default, DocumentBuilder creates a DOM that includes *all the lexical information necessary to reconstruct the original document, in it's original form*. That includes comment nodes as well as text nodes. There is as yet no standard mechanism for eliminating such lexical information in the DOM so you are left with the logical structure.

Moving down to the second `slide` element and opening the `item` element under it brings up the display shown in Figure 5.



```
    Comment: SUBSTITUTIONS WORK IN A
Element: slideshow
    Text:
    Comment: PROCESSING INSTRUCTI
    Text:
    ProcInstr: my.presentation.Program:, Q
    Text:
    Comment: TITLE SLIDE
    Text:
  Element: slide
    Text:
    Comment: TITLE SLIDE
    Text:
  Element: slide
      Text:
    Element: item
      EntityRef: copyright
          Comment: A SAMPLE copyr
          Text: This is the standard co
      Text:
    Text:
```

Figure 5: Entity Reference Node Displayed

Here, the Entity Reference node is highlighted. Note that the entity reference contains multiple nodes under it. This example shows only comment and a text nodes, but the entity could conceivable contain other element nodes, as well.

Moving down to the last `item` element under the last `slide` brings up the display shown in Figure 6.

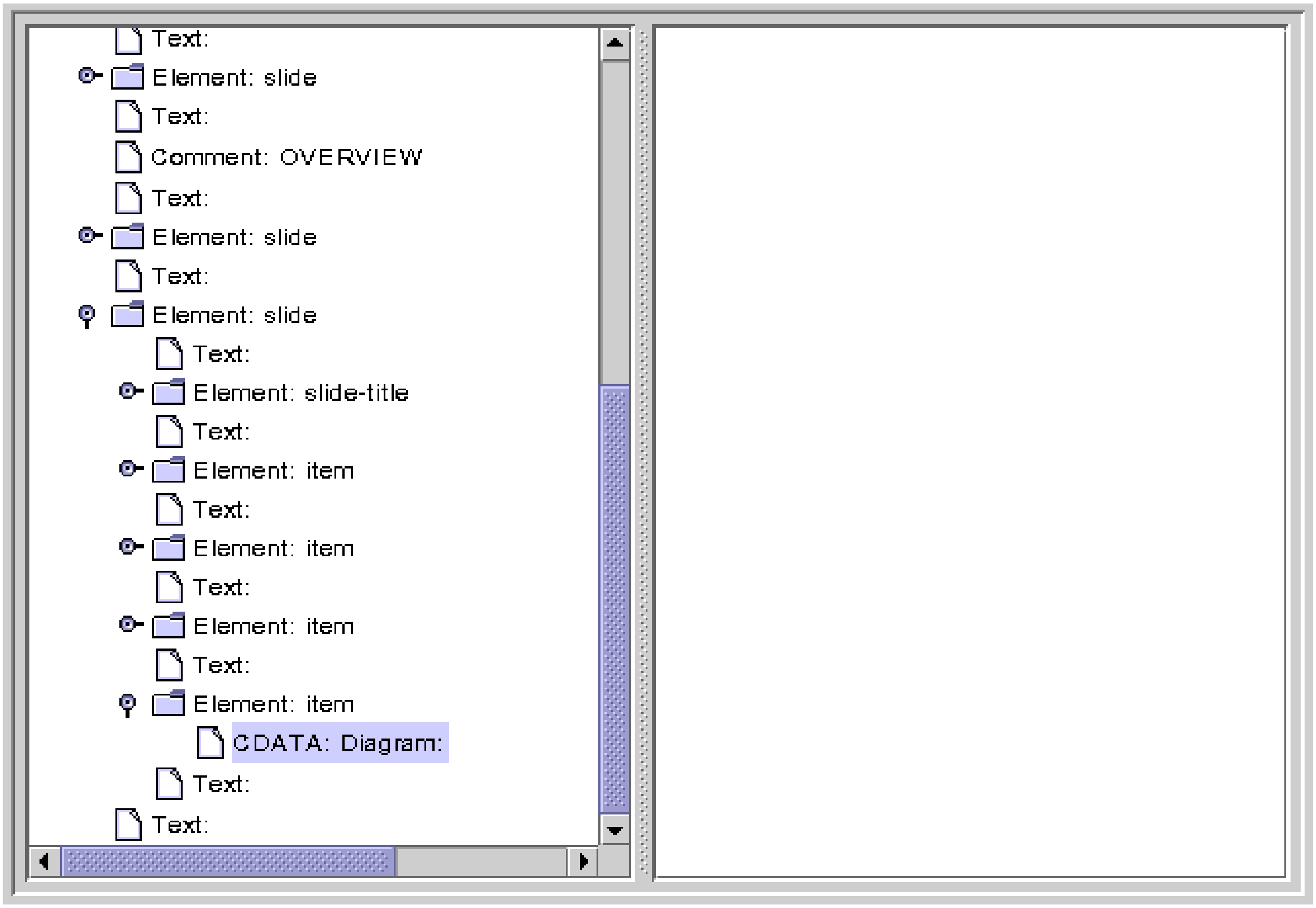


Figure 6: CDATA Node Displayed

Here, the CDATA node is highlighted. Note that there are no nodes under it. Since a CDATA section is entirely uninterpreted, all of its contents are contained in the node's `value` property.

Finishing Up

At this point, you have seen most of the nodes you will ever encounter in a DOM tree. There are one or two more that we'll mention in the next section, but you now know what you need to know to create or modify a DOM structure. In the next section, you'll see how to convert a DOM into a JTree that is suitable for an interactive GUI. Or, if you prefer, you can skip ahead to the 5th section of the DOM tutorial, Creating and Manipulating a DOM, where you'll learn how to create a DOM from scratch.

3. Constructing a User-Friendly JTree from a DOM

Now that you know what a DOM looks like internally, you'll be better prepared to modify a DOM or construct one from scratch . Before going on to that, though, this section presents some modifications to the JTreeModel that let you produce a more user-friendly version of the JTree suitable for use in a GUI.

Link Summary

Local Links

- Referencing Binary Entities
- Using the DTDHandler and EntityResolver

Exercise Links

- DomEcho03.java
- DomEcho04.java
- slideSample01.xml
- slideSample01-xml.html
- slideSample10.xml
- slideSample10-xml.html

External Links

- Understanding the TreeModel

Glossary Terms

well-formed

Compressing the Tree View

Displaying the DOM in tree form is all very well for experimenting and to learn how a DOM works. But it's not the kind of "friendly" display that most users want to see in a JTree. However, it turns out that very few modifications are needed to turn the TreeModel adapter into something that *will* present a user-friendly display. In this section, you'll make those modifications.

> **Note:**
> The code discussed in this section is in DomEcho03.java. The file it operates on is slideSample01.xml. (The browsable version is slideSample01-xml.html.)

Make the Operation Selectable

When you modify the adapter, you're going to *compress* the view of the DOM, eliminating all but the nodes you really want to display. Start by defining a boolean variable that controls whether you want the compressed or uncompressed view of the DOM:

```
public class DomEcho extends JPanel
{
    static Document document;


    Boolean compress = true;

    static final int windowHeight = 460;
    ...
```

Identify "Tree" Nodes

The next step is to identify the nodes you want to show up in the tree. To do that, go to the area where you defined the names of all the element types (in the `typeName` array), and add the code highlighted below:

```
public class DomEcho extends JPanel
{
    ...

    public static void makeFrame() {
        ...
    }

    // An array of names for DOM node-types
    static String[] typeName = {
        ...
    };
    final int ELEMENT_TYPE =   1;

    // The list of elements to display in the tree
    static String[] treeElementNames = {
        "slideshow",
        "slide",
        "title",         // For slideshow #1
        "slide-title",   // For slideshow #10
        "item",
    };
    Boolean treeElement(String elementName) {
      for (int i=0; i<treeElementNames.length; i++) {
        if ( elementName.equals(treeElementNames[i]) ) return true;
      }
      return false;
    }
```

With this code, you set up a constant you can use to identify the `ELEMENT` node type, declared the names of the elements you want in the tree, and created a method tells whether or not a given element name is a "tree element". Since slideSample01.xml has `title` elements and slideSample10.xml has `slide-title` elements, you set up the contents of this arrays so it would work with either data file.

> **Note:**
> The mechanism you are creating here depends on the fact that *structure* nodes like `slideshow` and `slide` never contain text, while text usually does appear in *content* nodes like `item`. Although those "content" nodes may contain subelements in slideShow10.xml, the DTD constrains those subelements to be XHTML nodes. Because they are XHTML nodes (an XML version of HTML that is constrained to be well-formed), the entire substructure under an `item` node can be combined into a single string and displayed in the `htmlPane` that makes up the other half of the application window. In the second part of this section, you'll do that concatenation, displaying the text and XHTML as content in the `htmlPane`.

Control Node Visibility

The next step is to modify the AdapterNode's `childCount` function so that it only counts "tree element" nodes -- nodes which are designated as displayable in the JTree. Make the modifications highlighted below to do that:

```
public class DomEcho extends JPanel
{
    ...
    public class AdapterNode
    {
      ...

      public AdapterNode child(int searchIndex) {
        ...
      }

      public int childCount() {
        if (!compress) {
          // Indent this
          return domNode.getChildNodes().getLength();
        }
        int count = 0;
        for (int i=0; i<domNode.getChildNodes().getLength(); i++) {
           org.w3c.dom.Node node = domNode.getChildNodes().item(i);
           if (node.getNodeType() == ELEMENT_TYPE
           && treeElement( node.getNodeName() ))
           {
             ++count;
           }
        }
        return count;
      }
    } // AdapterNode
```

The only tricky part about this code is checking to make sure the node is an element node before comparing the node. The DocType node makes that necessary, because it has the same name, "slideshow", as the `slideshow` element.

Control Child Access

Finally, you need to modify the AdapterNode's `child` function to return the Nth item from the list of displayable nodes, rather than the Nth item from all nodes in the list. Add the code highlighted below to do that:

```
public class DomEcho extends JPanel
{
    ...
    public class AdapterNode
    {
      ...

      public int index(AdapterNode child) {
        ...
```

```
          }

          public AdapterNode child(int searchIndex) {
            //Note: JTree index is zero-based.
            org.w3c.dom.Node node = domNode.getChildNodes()Item(searchIndex);
            if (compress) {
              // Return Nth displayable node
              int elementNodeIndex = 0;
              for (int i=0; i<domNode.getChildNodes().getLength(); i++) {
                node = domNode.getChildNodes()Item(i);
                if (node.getNodeType() == ELEMENT_TYPE
                && treeElement( node.getNodeName() )
                && elementNodeIndex++ == searchIndex) {
                   break;
                }
              }
            }
            return new AdapterNode(node);
          } // child

      }  // AdapterNode
```

There's nothing special going on here. It's a slightly modified version the same logic you used when returning the child count.

Check the Results

When you compile and run this version of the app on `slideSample01.xml`, and then expand the nodes in the tree, you see the results shown in Figure 1. The only nodes remaining in the tree are the high-level "structure" nodes.

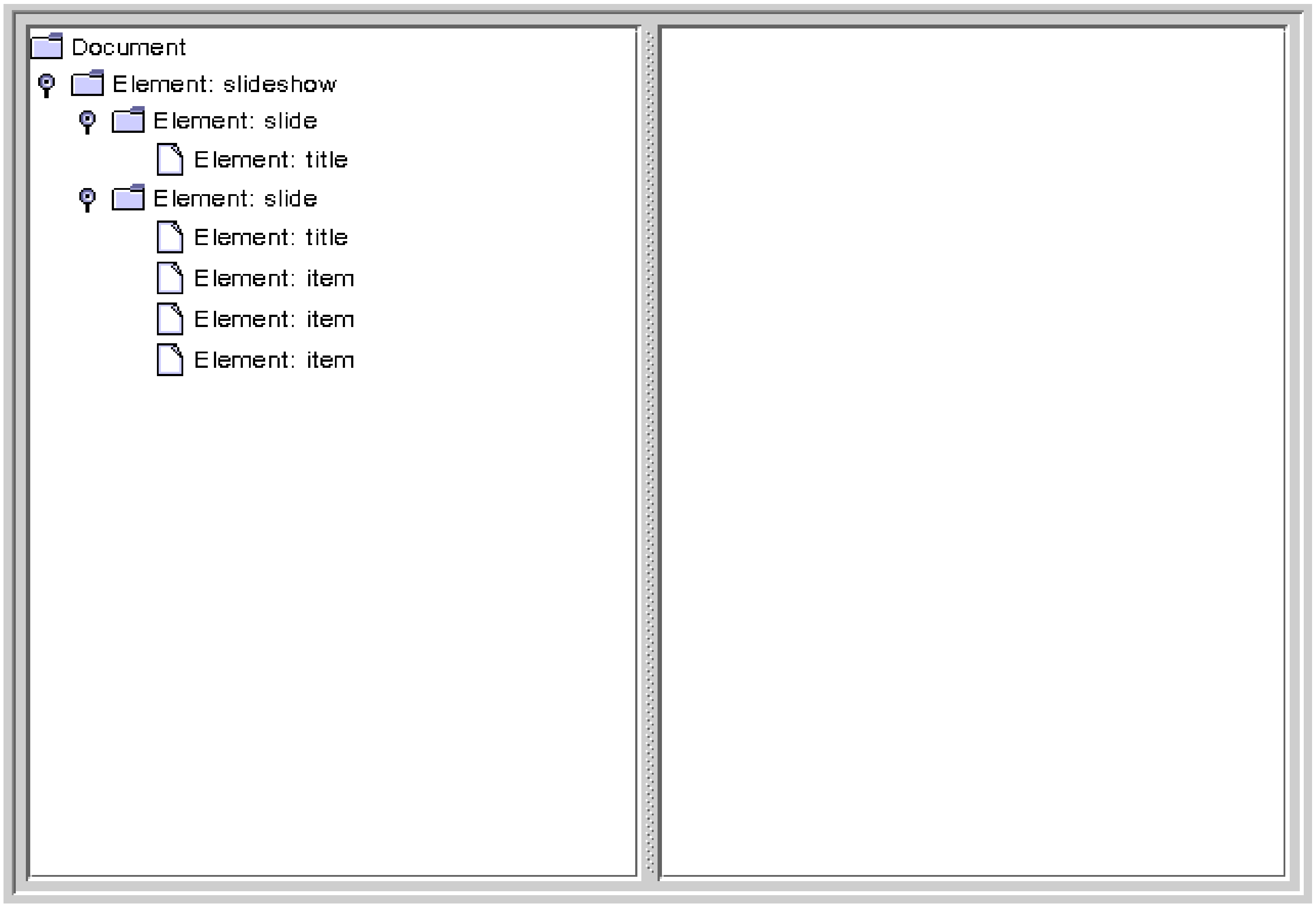


Figure 1: Tree View with a Collapsed Hierarchy

Extra Credit

The way the app stands now, the information that tells the app how to compress the tree for display is "hard coded". Here are some ways you could consider extending the app:

Use a Command-Line Argument

Whether you compress or don't compress the tree could be determined by a command line argument, rather than being a hard-coded Boolean variable. On the other hand, the list the list of elements that goes into the tree is still hard coded, so maybe that option doesn't make much sense, unless...

Read the treeElement list from a file

If you read the list of elements to include in the tree from an external file, that would make the whole app command driven. That would be good. But wouldn't it be really nice to derive that information from the DTD or schema, instead? So you might want to consider...

Automatically Build the List

Watch out, though! As things stand right now, there are no standard DTD parsers! If you use a DTD, then, you'll need to write your parser to make sense out of its somewhat arcane syntax. You'll probably have better luck if you use a schema, instead of a DTD. The nice thing about schemas is that use XML syntax, so you can use an XML parser to read the schema the same way you use any other file.

As you analyze the schema, note that the JTree-displayable *structure* nodes are those that have no text, while the *content* nodes may contain text and, optionally, XHTML subnodes. That distinction works for this example, and will likely work for a large body of real-world applications. It's pretty easy to construct cases that will create a problem, though, so you'll have to be on the lookout for schema/DTD specifications that embed non-XHTML elements in text-capable nodes, and take the appropriate action.

Acting on Tree Selections

Now that the tree is being displayed properly, the next step is to concatenate the subtrees under selected nodes to display them in the `htmlPane`. While you're at it, you'll use the concatenated text to put node-identifying information back in the JTree.

Note:
The code discussed in this section is in `DomEcho04.java`.

Identify Node Types

When you concatenate the sub nodes under an element, the processing you do is going to depend on the type of node. So the first thing to is to define constants for the remaining node types. Add the code highlighted below to do that:

```
public class DomEcho extends JPanel
{
    ...
    // An array of names for DOM node-types
    static String[] typeName = {
        ...
    };
    static final int ELEMENT_TYPE =   1;
    static final int ATTR_TYPE =      2;
    static final int TEXT_TYPE =      3;
    static final int CDATA_TYPE =     4;
    static final int ENTITYREF_TYPE = 5;
    static final int ENTITY_TYPE =    6;
    static final int PROCINSTR_TYPE = 7;
    static final int COMMENT_TYPE =   8;
    static final int DOCUMENT_TYPE =  9;
    static final int DOCTYPE_TYPE =  10;
    static final int DOCFRAG_TYPE =  11;
    static final int NOTATION_TYPE = 12;
```

Concatenate Subnodes to Define Element Content

Next, you need to define add the method that concatenates the text and subnodes for an element and returns it as the element's "content". To define the `content` method, you'll need to add the big chunk of code highlighted below, but this is the last big chunk of code in the DOM tutorial!.

```
public class DomEcho extends JPanel
{
```

```
...
public class AdapterNode
{
  ...
  public String toString() {
    ...
  }

  public String content() {
    String s = "";
    org.w3c.dom.NodeList nodeList = domNode.getChildNodes();
    for (int i=0; i<nodeList.getLength(); i++) {
      org.w3c.dom.Node node = nodeList.item(i);
      int type = node.getNodeType();
      AdapterNode adpNode = new AdapterNode(node);
      if (type == ELEMENT_TYPE) {
        if ( treeElement(node.getNodeName()) ) continue;
        s += "<" + node.getNodeName() + ">";
        s += adpNode.content();
        s += "</" + node.getNodeName() + ">";
      } else if (type == TEXT_TYPE) {
        s += node.getNodeValue();
      } else if (type == ENTITYREF_TYPE) {
        // The content is in the TEXT node under it
        s += adpNode.content();
      } else if (type == CDATA_TYPE) {
        StringBuffer sb = new StringBuffer( node.getNodeValue() );
        for (int j=0; j<sb.length(); j++) {
          if (sb.charAt(j) == '<') {
            sb.setCharAt(j, '&');
            sb.insert(j+1, "lt;");
            j += 3;
          } else if (sb.charAt(j) == '&') {
            sb.setCharAt(j, '&');
            sb.insert(j+1, "amp;");
            j += 4;
          }
        }
        s += "<pre>" + sb + "\n</pre>";
      }
    }
    return s;
  }
  ...
} // AdapterNode
```

This is not the most efficient code anyone ever wrote, but it works and will do fine for our purposes. In this code, you are recognizing and dealing with the following data types:

Element

For elements with names like the XHTML "em" node, you return the node's content sandwiched between the appropriate <em> and </em> tags. However, when processing the content for the `slideshow` element, for example, you don't include tags for the `slide` elements it contains so, when returning a node's content, you skip any subelements that are themselves displayed in the tree.

Text

No surprise here. For a text node, you simply return the node's `value`.

Entity Reference

Unlike CDATA nodes, Entity References can contain multiple subelements. So the strategy here is to return the concatenation of those subelements.

CDATA

Like a text node, you return the node's `value`. However, since the text in this case may contain angle brackets and ampersands, you need to convert them to a form that displays properly in an HTML pane. Unlike the XML CDATA tag, the HTML <pre> tag does preclude the parsing of character-format tags, break tags and the like. So you have to convert left-angle brackets (<) and ampersands (&) to get them to display properly.

On the other hand, there are quite a few node types you are *not* processing with the code above. It's worth a moment to examine them and understand why:

Attribute

These nodes do not appear in the DOM, but are obtained by invoking `getAttributes` on element nodes.

Entity

These nodes also do not appear in the DOM. They are obtained by invoking `getEntities` on DocType nodes.

Processing Instruction

These nodes don't contain displayable data.

Comment

Ditto. Nothing you want to display here.

Document

This is the root node for the DOM. There's no data to display for that.

DocType

The DocType node contains the DTD specification, with or without external pointers. It only appears under the root node, and has no data to display in the tree.

Document Fragment

This node is equivalent to a document node. It's a root node that the DOM specification intends for holding intermediate results during cut/paste operations, for example. Like a document node, there's no data to display.

Notation

We're just flat out ignoring this one. These nodes are used to include binary data in the DOM. As discussed earlier in Referencing Binary Entities and Using the DTDHandler and EntityResolver, the MIME types (in conjunction with namespaces) make a better mechanism for that.

Display the Content in the JTree

With the content-concatenation out of the way, only a few small programming steps remain. The first is to modify `toString` so that it uses the node's content for identifying information. Add the code highlighted below to do that:

```
public class DomEcho extends JPanel
{
    ...
    public class AdapterNode
    {
      ...
      public String toString() {
        ...
        if (! nodeName.startsWith("#")) {
           s += ": " + nodeName;
        }
        if (compress) {
           String t = content().trim();
           int x = t.indexOf("\n");
           if (x >= 0) t = t.substring(0, x);
           s += " " + t;
           return s;
        }
        if (domNode.getNodeValue() != null) {
           ...
        }
        return s;
      }
```

Wire the JTree to the JEditorPane

Returning now to the app's constructor, create a tree selection listener and use to wire the JTree to the JEditorPane:

```
public class DomEcho extends JPanel
{
    ...
    public DomEcho()
    {
       ...


       // Build right-side view
       JEditorPane htmlPane = new JEditorPane("text/html","");
       htmlPane.setEditable(false);
       JScrollPane htmlView = new JScrollPane(htmlPane);
       htmlView.setPreferredSize(
           new Dimension( rightWidth, windowHeight ));
```

```
          tree.addTreeSelectionListener(
            new TreeSelectionListener() {
              public void valueChanged(TreeSelectionEvent e) {
                TreePath p = e.getNewLeadSelectionPath();
                if (p != null) {
                  AdapterNode adpNode =
                     (AdapterNode) p.getLastPathComponent();
                  htmlPane.setText(adpNode.content());
                }
              }
            }
          );
```

Now, when a JTree node is selected, it's contents are delivered to the `htmlPane`.

> **Note:**
> The TreeSelectionListener in this example is created using an anonymous inner-class adapter. If you are programming for the 1.1 version of the platform, you'll need to define an external class for this purpose.

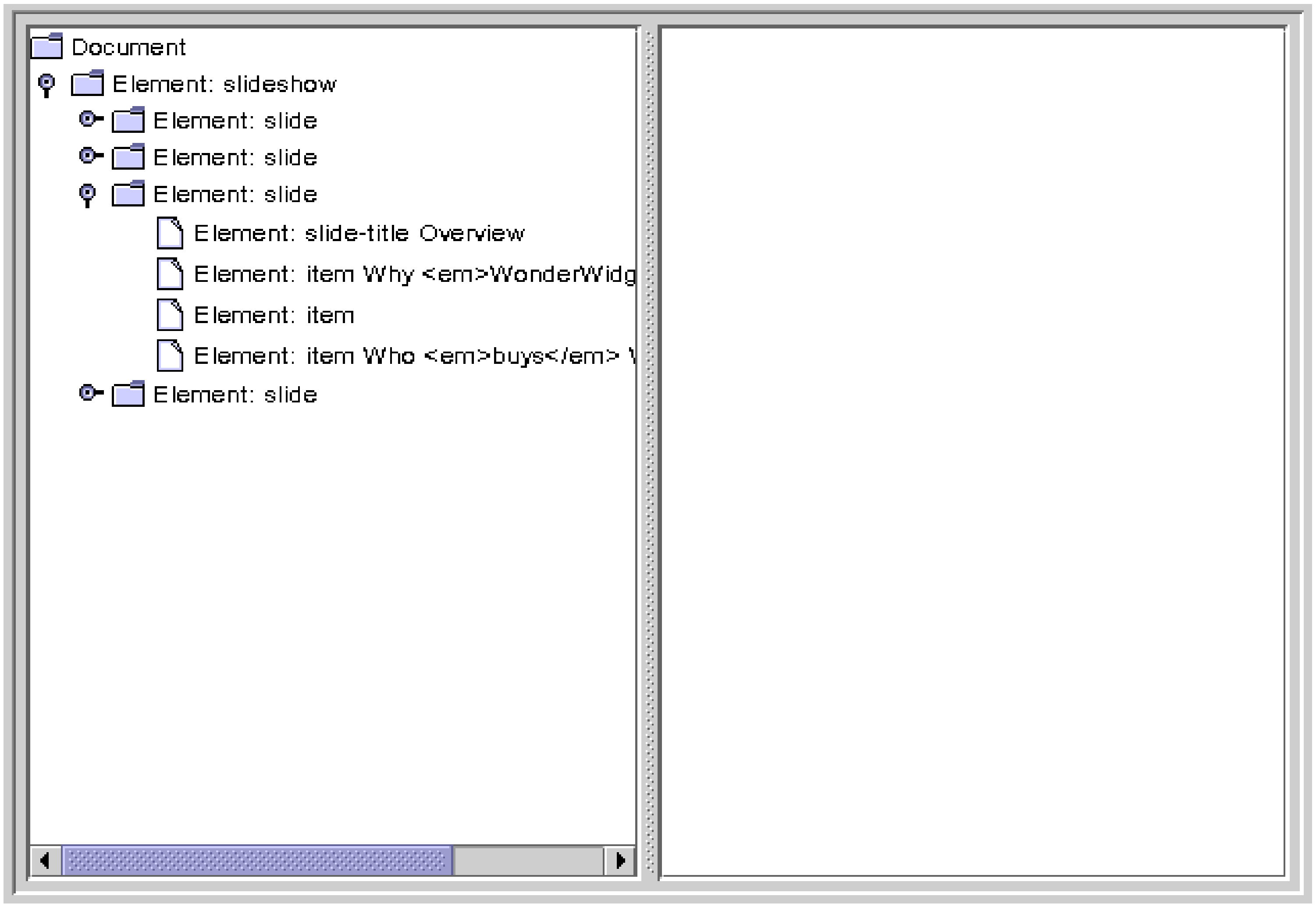
If you compile this version of the app, you'll discover immediately that the `htmlPane` needs to be specified as `final` to be referenced in an inner class, so add the keyword highlighted below:

```
public DomEcho04()
{
   ...

   // Build right-side view
   final JEditorPane htmlPane = new JEditorPane("text/html","");
   htmlPane.setEditable(false);
   JScrollPane htmlView = new JScrollPane(htmlPane);
   htmlView.setPreferredSize(
       new Dimension( rightWidth, windowHeight ));
```

Run the App

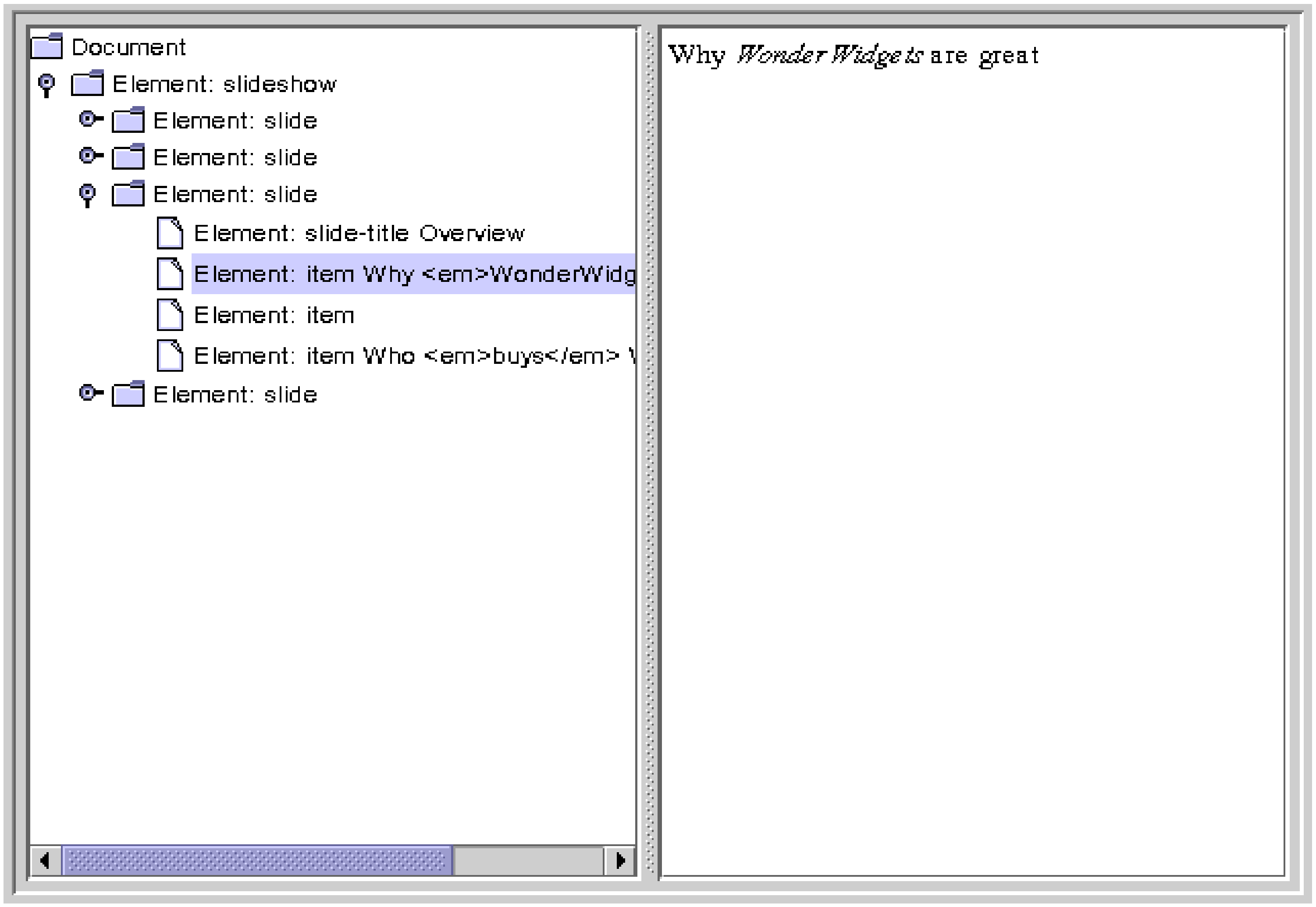
When you compile the app and run it on slideSample10.xml (the browsable version is slideSample10-xml.html), you get a display like that shown in Figure 2. Expanding the hierarchy shows that the JTree now includes identifying text for a node whenever possible.



```
Document
    Element: slideshow
        Element: slide
        Element: slide
        Element: slide
            Element: slide-title Overview
            Element: item Why <em>WonderWidg
            Element: item
            Element: item Who <em>buys</em> V
        Element: slide
```

Figure 2: Collapsed Hierarchy Showing Text in Nodes

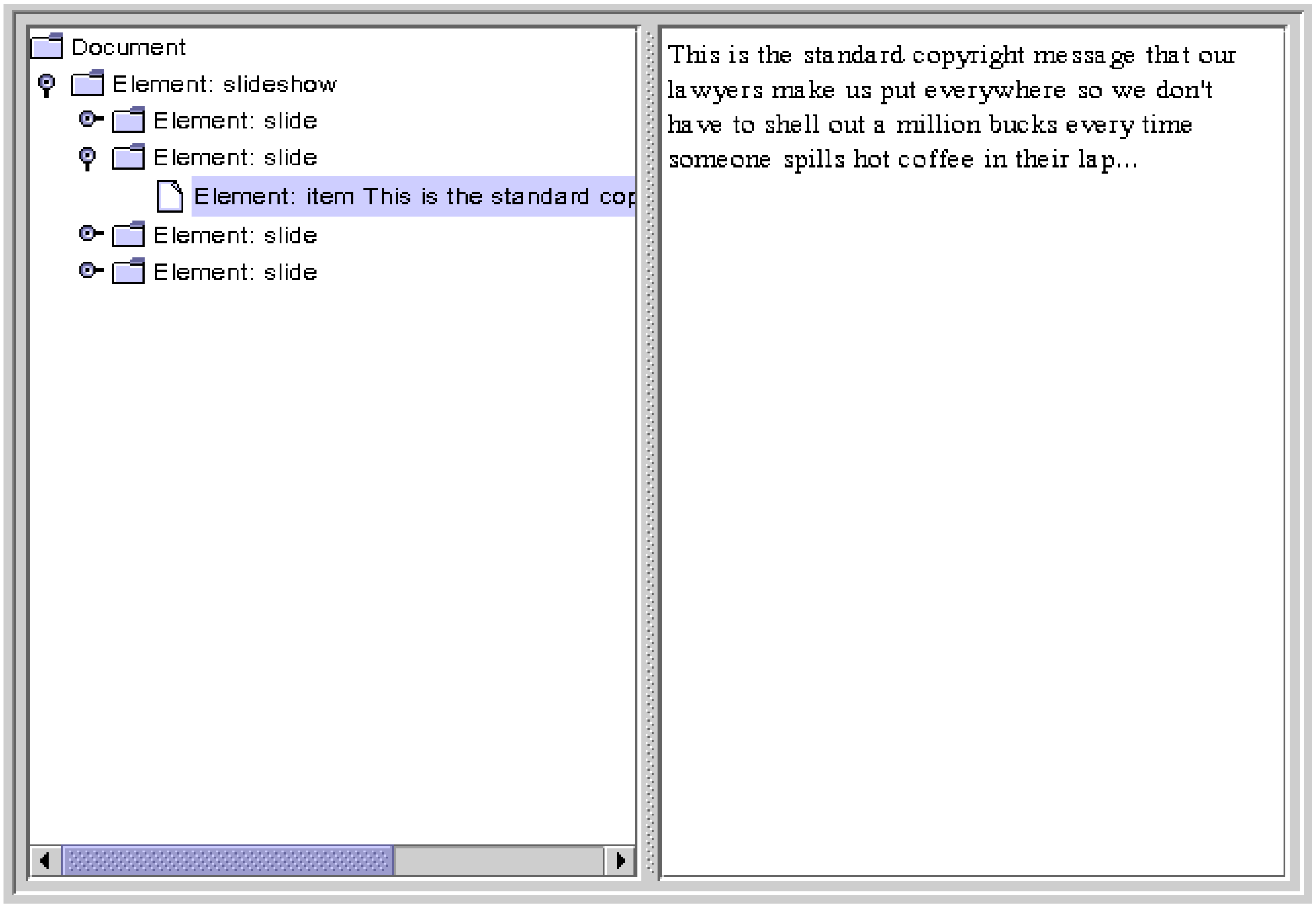
Selecting an item that includes XHTML subelements produces a display like that shown in Figure 3:



Why *Wonder Widgets* are great

Figure 3: Node with <em> Tag Selected

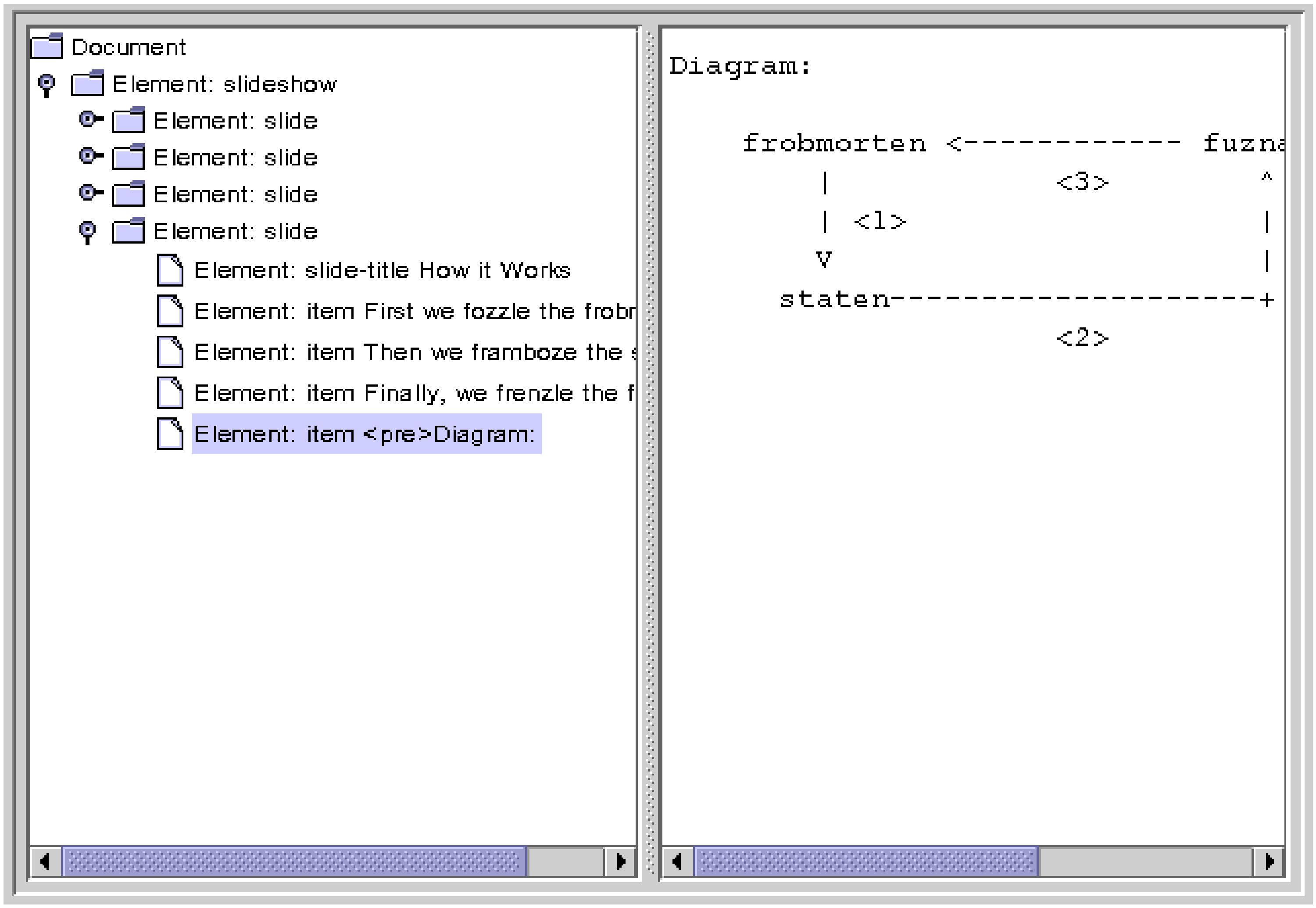
Selecting a node that contains an entity reference causes the entity text to be included, as shown in Figure 4:



This is the standard copyright message that our lawyers make us put everywhere so we don't have to shell out a million bucks every time someone spills hot coffee in their lap...

Figure 4: Node with Entity Reference Selected

Finally, selecting a node that includes a CDATA section produces results like those shown in Figure 5:


Document
Element: slideshow
Element: slide
Element: slide
Element: slide
Element: slide
Element: slide-title How it Works
Element: item First we fozzle the frobr
Element: item Then we framboze the s
Element: item Finally, we frenzle the f
Element: item <pre>Diagram:

```
Diagram:

    frobmorten <------------ fuzna
        |             <3>        ^
        | <1>                    |
        V                        |
      staten--------------------+
                      <2>
```

Figure 5: Node with CDATA Component Selected

Extra Credit

Now that you have the app working, here are some ways you might think about extending it in the future:

Use Title Text to Identify Slides
: Special case the `slide` element so that the contents of the `title` node is used as the identifying text. When selected, convert the title node's contents to a centered H1 tag, and ignore the `title` element when constructing the tree.

Convert Item Elements to Lists
: Remove `item` elements from the JTree and convert them to html lists using <ul>, <li>, </UL> tags, including them in the slide's content when the slide is selected.

Handling Modifications

A full discussion of the mechanisms for modifying the JTree's underlying data model is beyond the scope of this tutorial. However, a few words on the subject are in order.

Most importantly, note that if you allow the user to modifying the structure by manipulating the JTree, you have take the compression into account when you figure out where to apply the change. For example, if you are displaying text in the tree and the user modifies that, the changes would have to be applied to text subelements, and perhaps require a rearrangement of the XHTML subtree.

When you make those changes, you'll need to understand more about the interactions between a JTree, it's TreeModel, and an underlying data model. That subject is covered in depth in the Swing Connection article, Understanding the TreeModel.

Finishing Up

You now understand pretty much what there is know about the structure of a DOM, and you know how to adapt a DOM to create a user-friendly display in a JTree. It has taken quite a bit of coding, but in return you have obtained valuable tools for exposing a DOM's structure and a template for GUI apps. In the next section, you'll make a couple of minor modifications to the code that turn the app into a vehicle for experimentation, and then experiment with building and manipulating a DOM.

Top Contents Index Glossary

4. Creating and Manipulating a DOM

By now, you understand the structure of the nodes that make up a DOM. A DOM is actually very easy to create. This section of the DOM tutorial is going to take much less work than anything you've see up to now. All the foregoing work, however, generated the basic understanding that will make this section a piece of cake.

Link Summary
Exercise Links • DomEcho05.java • DomEcho06.java ***API Links*** • org.w3c.dom.Node • org.w3c.dom.Element ***Glossary Terms*** normalization

Obtaining a DOM from the Factory

In this version of the application, you're still going to create a document builder factory, but this time you're going to tell it create a new DOM instead of parsing an existing XML document. You'll keep all the existing functionality intact, however, and add the new functionality in such a way that you can "flick a switch" to get back the parsing behavior.

Note:
The code discussed in this section is in `DomEcho05.java`.

Modify the Code

Start by turning off the compression feature. As you work with the DOM in this section, you're going to want to see all the nodes.

```
public class DomEcho05  extends JPanel
{
    ...
    boolean compress = true;
    boolean compress = false;
```

Next, you need to create a `buildDom` method that creates the `document` object. The easiest way to do that is to create the method and then copy the DOM-construction section from the `main` method to create the `buildDom`. The modifications shown below show you the changes you need to make to make that code suitable for the `buildDom` method.

```
public class DomEcho05  extends JPanel
{
    ...
    public static void makeFrame() {
        ...
    }
```

```
        public static void buildDom()
        {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            try {
              DocumentBuilder builder = factory.newDocumentBuilder();
              document = builder.parse( new File(argv[0]) );
              document = builder.newDocument();  // Create from whole cloth

            } catch (SAXException sxe) {
               ...

            } catch (ParserConfigurationException pce) {
                // Parser with specified options can't be built
                pce.printStackTrace();

            } catch (IOException ioe) {
               ...
            }
        }
```

In this code, you replaced the line that does the parsing with one that creates a DOM. Then, since the code is no longer parsing an existing file, you removed exceptions which are no longer thrown: SAXException and IOException.

And since you are going to be working with Element objects, add the statement to import that class at the top of the program:

```
    import org.w3c.dom.Document;
    import org.w3c.dom.DOMException;
    import org.w3c.dom.Element;
```

Create Element and Text Nodes

Now, for your first experiment, add the Document operations to create a root node and several children:

```
    public class DomEcho05  extends JPanel
    {
        ...
        public static void buildDom()
        {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            try {
              DocumentBuilder builder = factory.newDocumentBuilder();
              document = builder.newDocument();  // Create from whole cloth

              Element root =
                      (Element) document.createElement("rootElement");
              document.appendChild(root);
              root.appendChild( document.createTextNode("Some") );
              root.appendChild( document.createTextNode(" ")    );
              root.appendChild( document.createTextNode("text") );
            } catch (ParserConfigurationException pce) {
                // Parser with specified options can't be built
                pce.printStackTrace();
            }
        }
```

Finally, modify the argument-list checking code at the top of the `main` method so you invoke `buildDom` and `makeFrame` instead of generating an error, as shown below:

```
    public class DomEcho05  extends JPanel
    {
        ...
        public static void main(String argv[])
        {
            if (argv.length != 1) {
                System.err.println("Usage: java DomEcho filename");
                System.exit(1);
                buildDom();
                makeFrame();
                return;
            }
```

That's all there is to it! Now, if you supply an argument the specified file is parsed and, if you don't, the experimental code that builds a DOM is executed.

Run the App

Compile and run the program with no arguments produces the result shown in Figure 1:

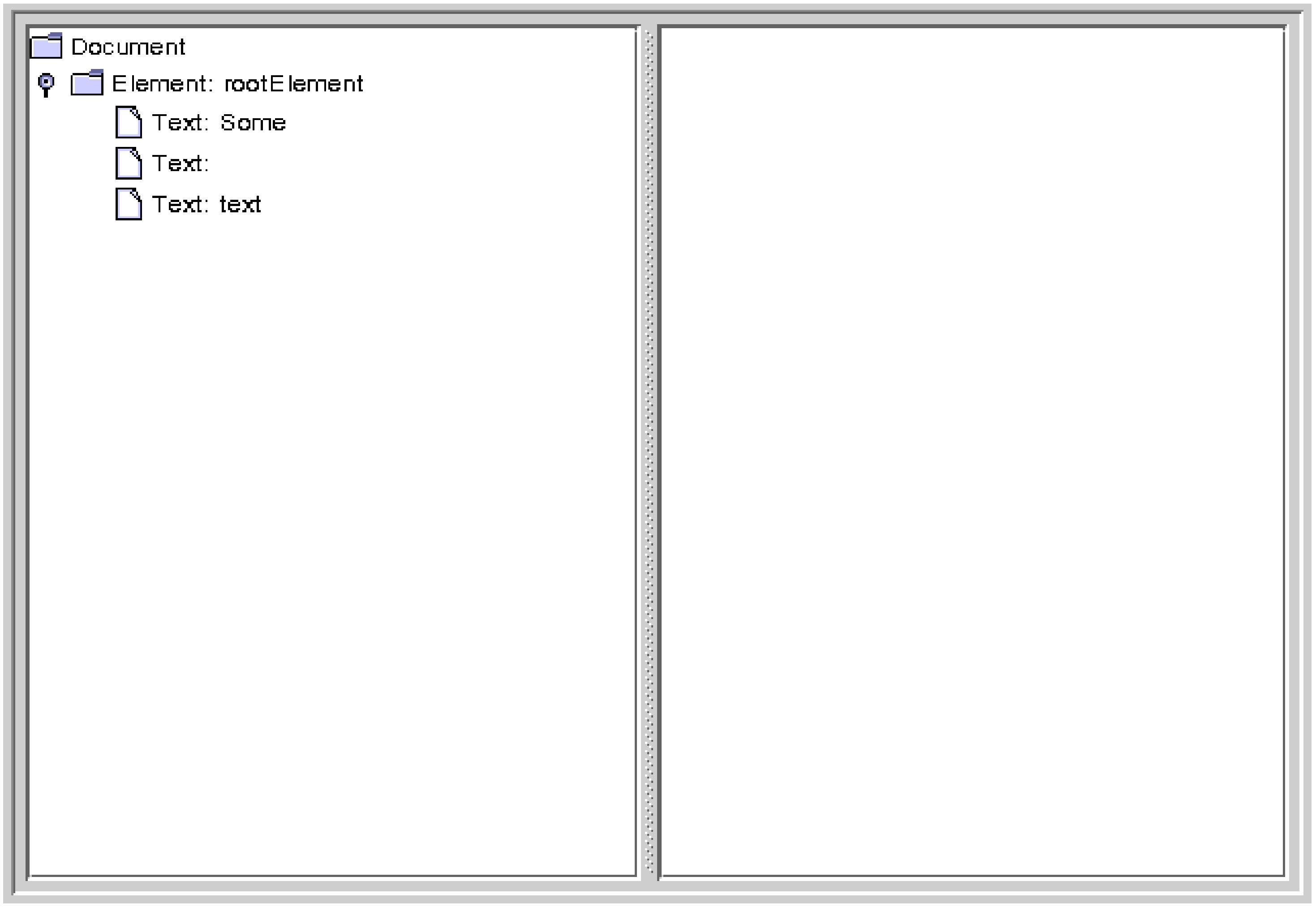


Figure 1: Element Node and Text Nodes Created

Normalizing the DOM

In this experiment, you'll manipulate the DOM you created by normalizing it (cf. normalization) after it has been constructed.

Note:
The code discussed in this section is in `DomEcho06.java`.

Add the code highlighted below to normalize the DOM:.

```
public static void buildDom()
{
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    try {
      ...
      root.appendChild( document.createTextNode("Some") );
      root.appendChild( document.createTextNode(" ")    );
      root.appendChild( document.createTextNode("text") );

      document.getDocumentElement().normalize();

    } catch (ParserConfigurationException pce) {
```

```
                              ...
```

In this code, `getDocumentElement` returns the document's root node, and the `normalize` operation manipulates the tree under it.

When you compile and run the app now, the result looks like Figure 2:

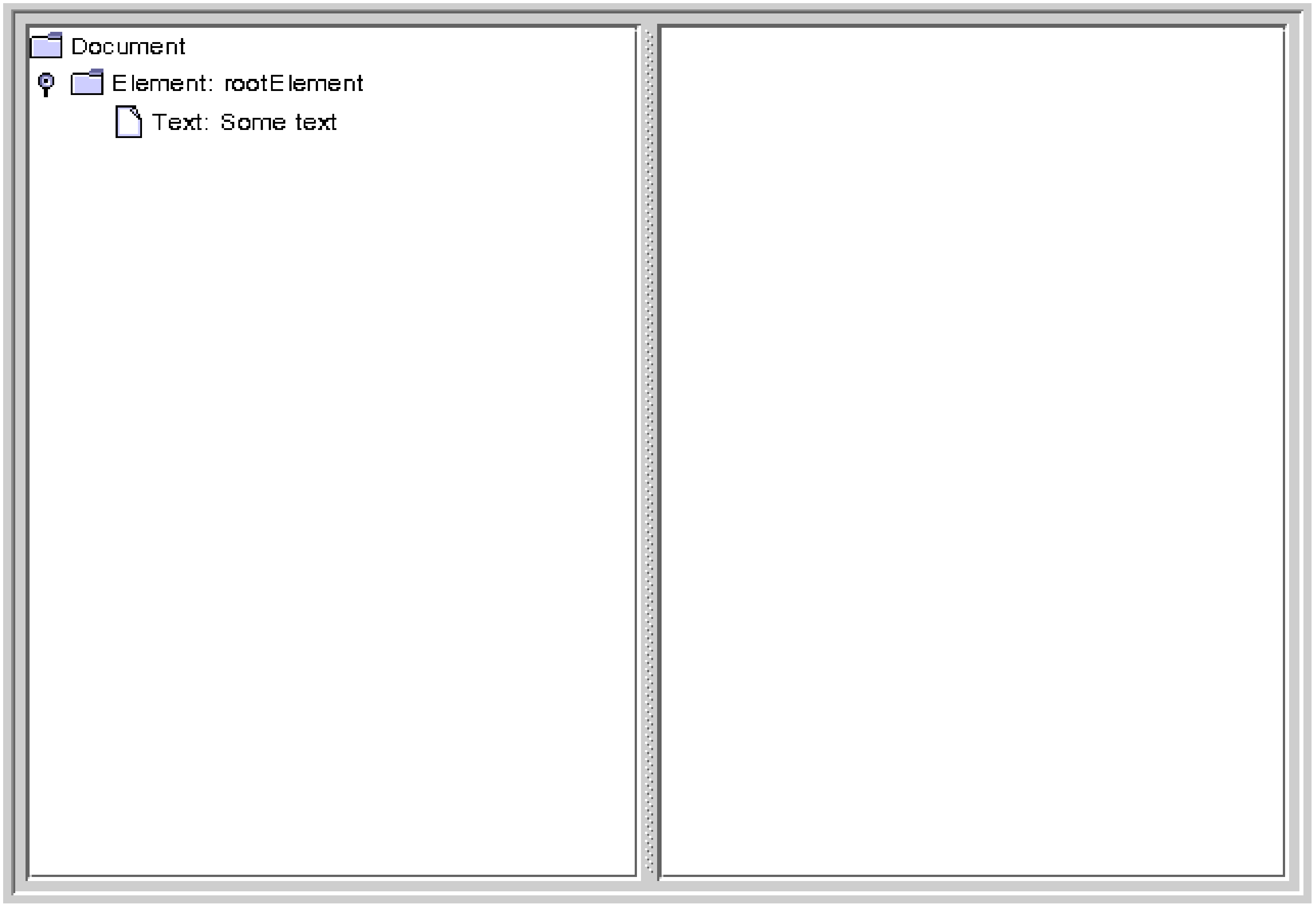


Figure 2: Text Nodes Merged After Normalization

Here, you can see that the adjacent text nodes have been combined into a single node. The normalize operation is one that you will typically want to use after making modifications to a DOM, to ensure that the resulting DOM is as compact as possible.

> **Note:**
> Now that you have this program to experiment with, see what happens to other combinations of CDATA, entity references, and text nodes when you normalize the tree.

Other Operations

To complete this section, we'll take a quick look at some of the other operations you might want to apply to a DOM, including:\

- Traversing nodes

- Creating attributes
- Removing nodes

Traversing Nodes

The org.w3c.dom.Node interface defines a number of methods you can use to traverse nodes, including `getFirstChild`, `getLastChild`, `getNextSibling`, `getPreviousSibling`, and `getParentNode`. Those operations are sufficient to get from anywhere in the tree to any other location in the tree.

Creating Attributes

The org.w3c.dom.Element interface, which extends Node, defines a `setAttribute` operation, which adds an attribute to that node. (A better name from the Java platform standpoint would have been `addAttribute`, since the attribute is not a property of the class, and since a new object is created.)

You can also use the Document's `createAttribute` operation to create an instance of Attribute, and use an overloaded version of `setAttribute` to add that.

Removing and Changing Nodes

To remove a node, you use its parent Node's `removeChild` method. To change it, you can either use the parent node's `replaceChild` operation or the node's `setNodeValue` operation.

Finishing Up

Congratulations! You've learned how a DOM is structured and how to manipulate it. And you now have a DomEcho application that you can use to display a DOM's structure, condense it down to GUI-compatible dimensions, and experiment with to see how various operations affect the structure. Have fun with it!

Top Contents Index Glossary

5. Using Namespaces

Link Summary

Local Links

- Defining Attributes in the DTD

External Links

- Namespace Specification

Glossary Terms

attribute, namespace, URI, URL, URN

As you saw previously, one way or another it is necessary to resolve the conflict between the `title` element defined in **slideshow.dtd** and the one defined in **xhtml.dtd**. In the previous exercise, you hyphenated the name in order to put it into a different "namespace". In this section, you'll see how to use the XML namespace standard to do the same thing without renaming the element.

> **Note:** At this point in time, the Java XML parsers do not support namespaces. This section is for information only.

The primary goal of the namespace specification is to let the document author tell the parser which DTD to use when parsing a given element. The parser can then consult the appropriate DTD for an element definition. Of course, it is also important to keep the parser from aborting when a "duplicate" definition is found, and yet still generate an error if the document references an element like `title` without *qualifying* it (identifying the DTD to use for the definition).

> **Note:**
> Namespaces apply to attributes as well as to elements. In this section, we consider only elements. For more information on attributes, consult the namespace specification at http://www.w3.org/TR/REC-xml-names/.

Defining a Namespace

To define a namespace that an element belongs to, it is necessary to add an attribute to the element's definition, where the attribute name is `xmlns` ("xml namespace"). For example, you could do that in **slideshow.dtd** by adding an entry like the following in the `title` element's attribute-list definition:

```
<!ELEMENT title (%inline;)*>
<!ATTLIST title
   xmlns CDATA #FIXED "http://www.example.com/slideshow"
>
```

Declaring the attribute as `FIXED` has several important features:

- It prevents the document from specifying any non-matching value for the `xmlns` attribute (as described in Defining Attributes in the DTD).
- The element defined in this DTD is made unique (because the parser understands the `xmlns` attribute), so it does not conflict with an element that has the same name in another DTD. That allows multiple DTDs to use the same element name without generating a parser error.
- When a document specifies the `xmlns` attribute for a tag, the document selects the element definition with a matching attribute.

To be thorough, every element name in your DTD would get the exact same attribute, with the same value. (Here, though, we're only concerned about the `title` element.) Note, too, that you are using a `CDATA` string to supply the URI. In this case, we've specified an URL. But you could also specify a URN, possibly by specifying a prefix like `urn:` instead of `http:`. (URNs are currently being researched. They're not seeing a lot of action at the moment, but that could change in the future.)

Referencing a Namespace

When a document uses an element name that exists in only one of the `.dtd` files it references, the name does not need to be qualified. But when an element name that has multiple definitions is used, some sort of qualification is a necessity.

> **Note:**
> In point of fact, an element name is always qualified by it's *default namespace*, as defined by name of the DTD file it resides in. As long as there as is only one definition for the name, the qualification is implicit.

You qualify a reference to an element name by specifying the xmlns attribute, as shown here:

```
<title xmlns="http://www.example.com/slideshow"
    Overview
</title>
```

The specified namespace applies to that element, and to any elements contained within it.

Defining a Namespace Prefix

When you only need one namespace reference, it's not such a big deal. But when you need to make the same reference several times, adding `xmlns` attributes becomes unwieldy. It also makes it harder to change the name of the namespace at a later date.

The alternative is to define a *namespace prefix*, which as simple as specifying xmlns, a colon (:) and the prefix name before the attribute value, as shown here:

```
<sl:slideshow xmlns:SL='http:/www.example.com/slideshow'
              ...>
    ...
</SL:slideshow>
```

This definition sets up `SL` as a prefix that can be used to qualify the current element name and any element within it. Since the prefix can be used on any of the contained elements, it makes the most sense to define it on the XML document's root element, as shown here.

> **Note:**
> The namespace URI can contain characters which are not valid in an XML name, so it cannot be used as a prefix directly. The prefix definition associates an XML name with the URI, which allows the prefix name to be used instead. It also makes it easier to change references to the URI in the future.

When the prefix is used to qualify an element name, the end-tag also includes the prefix, as highlighted here:

```
<SL:slideshow xmlns:SL='http:/www.example.com/slideshow'
              ...>
    ...
    <slide>
       <SL:title>Overview<SL:title>
    </slide>
    ...
</SL:slideshow>
```

Finally, note that multiple prefixes can be defined in the same element, as shown here:

```
<SL:slideshow xmlns:SL='http:/www.example.com/slideshow'
              xmlns:xhtml='urn:...'>
    ...
    </SL:slideshow>
```

With this kind of arrangement, all of the prefix definitions are together in one place, and you can use them anywhere they are needed in the document. This example also suggests the use of URN to define the `xhtml` prefix, instead of an URL. That definition would conceivably allow the app to reference a local copy of the XHTML DTD or some mirrored version, with a potentially beneficial impact on performance..

---

3. Generating XML from an Arbitrary Data Structure

In this section, you'll use an XSLT transformer to converting an *arbitrary data structure* to XML.

In general outline, then, you're going to:

a. Modify an existing program that reads the data and modify it to generate SAX events. (Whether that is a real parser or simply a data filter of some kind is irrelevant for the moment.)
2. You'll then use the SAX "parser" to construct a SAXSource for the transformation.
3. You'll use the same StreamResult object you created in the last exercise, so you can see the results. (But note that you could just as easily create a DOMResult object to create a DOM in memory.)
4. You'll wire the source to the result, using the XSLT transformer object to make the conversion.

For starters, you need a data set you want to convert and some program which is capable of reading the data. In the next two sections, you'll create a simple data file and a program that reads it.

Link Summary

Exercise Links

- PersonalAddressBook.ldif
- AddressBookReader01.java
- AddressBookReaderLog01
- TransformationApp04.java
- TransformationLog04

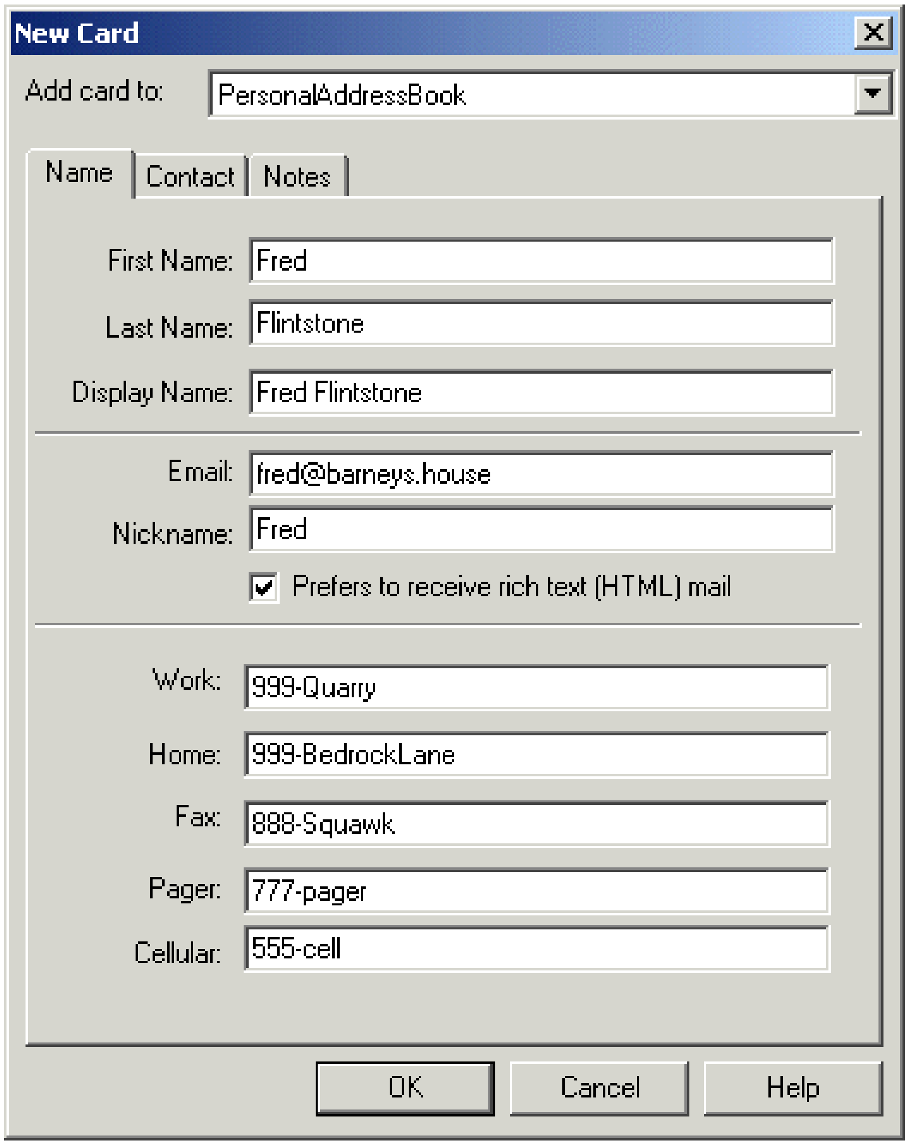
API Links

- ContentHandler
- InputSource
- SAXSource
- XmlReader

Creating A Simple File

We'll start by creating a data set for an address book. You can duplicate the process, if you like, or simply make use of the data stored in PersonalAddressBook.ldif.

The file shown below were produced by creating a new address book in Netscape messenger, giving it some dummy data (one address card) and then exporting it in LDIF format. Here is the address book entry that was created:



Add card to: PersonalAddressBook

First Name: Fred
Last Name: Flintstone
Display Name: Fred Flintstone
Email: fred@barneys.house
Nickname: Fred
[✓] Prefers to receive rich text (HTML) mail
Work: 999-Quarry
Home: 999-BedrockLane
Fax: 888-Squawk
Pager: 777-pager
Cellular: 555-cell

Exporting the address book produces a file like the one shown below. The parts of the file that we care about are shown in bold.

```
dn: cn=Fred Flinstone,mail=fred@barneys.house
modifytimestamp: 20010409210816Z
cn: Fred Flinstone
xmozillanickname: Fred
mail: Fred@barneys.house
xmozillausehtmlmail: TRUE
givenname: Fred
sn: Flinstone
telephonenumber: 999-Quarry
homephone: 999-BedrockLane
facsimiletelephonenumber: 888-Squawk
pagerphone: 777-pager
cellphone: 555-cell
xmozillaanyphone: 999-Quarry
objectclass: top
objectclass: person
```

Note that: each line of the file contains a variable name, a colon, and a space followed by a value for the variable. The "sn" variable contains the person's surname (last name) and, for some reason, the variable "cn" contains the DisplayName field from the address book entry.

> **Note:**
> LDIF stands for LDAP Data Interchange Format, according to the Netscape pages. And LDAP, turn, stands for Lightweight Directory Access Protocol. I prefer to think of LDIF as the "Line Delimited Interchange Format", since that is pretty much what it is.

Creating A Simple Parser

The next step is to create a program that parses the data. Again, you can follow the process to write your own if you like, or simply make a copy of the program so you can use it to do the XSLT-related exercises that follow.

> **Note:**
> The code discussed in this section is in AddressBookReader01.java. The output is in AddressBookReaderLog01.

The text for the program is shown below. It's an absurdly simple program that doesn't even loop for multiple entries because, after all, it's just a demo!

```
import java.io.*;


public class AddressBookReader01
{

    public static void main(String argv[])
    {
        // Check the arguments
        if (argv.length != 1) {
            System.err.println ("Usage: java AddressBookReader filename");
            System.exit (1);
        }
        String filename = argv[0];
        File f = new File(filename);
        AddressBookReader01 reader = new AddressBookReader01();
        reader.parse(f);
    }

    /** Parse the input */
    public void parse(File f)
    {
        try {
            // Get an efficient reader for the file
            FileReader r = new FileReader(f);
```

```
                BufferedReader br = new BufferedReader(r);

                // Read the file and display it's contents.
                String line = br.readLine();
                while (null != (line = br.readLine())) {
                  if (line.startsWith("xmozillanickname: ")) break;
                }

                output("nickname", "xmozillanickname", line);
                line = br.readLine();
                output("email",    "mail",             line);
                line = br.readLine();
                output("html",     "xmozillausehtmlmail", line);
                line = br.readLine();
                output("firstname","givenname",        line);
                line = br.readLine();
                output("lastname", "sn",               line);
                line = br.readLine();
                output("work",     "telephonenumber",  line);
                line = br.readLine();
                output("home",     "homephone",        line);
                line = br.readLine();
                output("fax",      "facsimiletelephonenumber", line);
                line = br.readLine();
                output("pager",    "pagerphone",       line);
                line = br.readLine();
                output("cell",     "cellphone",        line);

            }
            catch (Exception e) {
                e.printStackTrace();
            }
        }

        void output(String name, String prefix, String line)
        {
            int startIndex = prefix.length() + 2;  // 2=length of ": "
            String text = line.substring(startIndex);
            System.out.println(name + ": " + text);
        }
    }
```

This program contains 3 methods:

main

The main method gets the name of the file from the command line, creates an instance of the parser, and sets it to work parsing the file. This method will be going away when we convert the program into a SAX parser. (That's one reason for putting the parsing code into a separate method.)

parse

This method operates on the File object sent to it by the main routine. As you can see, its about as simple as it can get! The only nod to efficiency is the use of a BufferedReader, which can become important when you start operating on large files.

output

The output method contains the smarts about the structure of a line. Starting from the right It takes 3 arguments. The first argument gives the method a name to display, so we can output "html" as a variable name, instead of "xmozillausehtmlmail". The second argument gives the variable name stored in the file (xmozillausehtmlmail). The third argument gives the line containing the data. The routine then strips off the variable name from the start of the line and outputs the desired name, plus the data.

Running this program on the address book file produces this output:

```
nickname: Fred
email: Fred@barneys.house
html: TRUE
firstname: Fred
lastname: Flintstone
work: 999-Quarry
home: 999-BedrockLane
fax: 888-Squawk
pager: 777-pager
cell: 555-cell
```

I think we can all agree that's a bit more readable!

Modifying the Parser to Generate SAX Events

The next step is to modify the parser to generate SAX events, so you can use it as the basis for a SAXSource object in an XSLT transform.

> **Note:**
> The code discussed in this section is in AddressBookReader02.java.

Start by extending importing the additional classes you're going to need:

```
import java.io.*;

import org.xml.sax.*;
Import org.xml.sax.helpers.AttributesImpl;
```

Next, modify the application so that it extends XmlReader. That converts the app into a parser that generates the appropriate SAX events.

```
Public class AddressBookReader02
  implements XMLReader
```

```
    {
```

Now, remove the `main` method. You won't be needing that any more.

```
    Public static void main(String argv[])
    {
        // Check the arguments
        if (argv.length != 1) {
            System.err.println ("Usage: Java AddressBookReader filename");
            System.exit (1);
        }
        String filename = argv[0];
        File f = new File(filename);
        AddressBookReader02 reader = new AddressBookReader02();
        reader.parse(f);
    }
```

Add some global variables that will come in handy in a few minutes:

```
        ContentHandler handler;

        // We're not doing namespaces, and we have no
        // attributes on our elements.
        String nsu = "";  // NamespaceURI
        Attributes atts = new AttributesImpl();
        String rootElement = "addressbook";

        String indent = "\n    "; // for readability!
```

The SAX ContentHandler is the thing that is going to get the SAX events the parser generates. To make the app into an XmlReader, you'll be defining a `setContentHandler` method. The `handler` variable will hold the result of that configuration step.

And, when the parser generates SAX *element* events, it will need to supply namespace and attribute information. Since this is a simple application, you're defining null values for both of those.

You're also defining a root element for the data structure (addressbook), and setting up an indent string to improve the readability of the output.

Next, modify the parse method so that it takes an InputSource as an argument, rather than a File, and account for the exceptions it can generate:

```
    public void parse(~~File f)~~InputSource input)
    throws IOException, SAXException
```

Now make the changes shown below to get the reader encapsulated by the InputSource object:

```
try {
    // Get an efficient reader for the file
    FileReader r = new FileReader(f);
    java.io.Reader r = input.getCharacterStream();
    BufferedReader Br = new BufferedReader(r);
```

Note:
In the next section, you'll create the input source object and what you put in it will, in fact, be a buffered reader. But the AddressBookReader could be used by someone else, somewhere down the line. This step makes sure that the processing will be efficient, regardless of the reader you are given.

The next step is to modify the `parse` method to generate SAX events for the start of the document and the root element. Add the code highlighted below to do that:

```
/** Parse the input */
public void parse(InputSource input)
...
{
    try {
        ...
        // Read the file and display it's contents.
        String line = br.readLine();
        while (null != (line = br.readLine())) {
          if (line.startsWith("xmozillanickname: ")) break;
        }

        if (handler==null) {
          throw new SAXException("No content handler");
        }
        handler.startDocument();
        handler.startElement(nsu, rootElement, rootElement, atts);

        output("nickname", "xmozillanickname", line);
        ...
        output("cell",     "cellphone",        line);

        handler.ignorableWhitespace("\n".toCharArray(),
                                    0, // start index
                                    1  // length
                                    );
        handler.endElement(nsu, rootElement, rootElement);
        handler.endDocument();
    }
    catch (Exception e) {
    ...
```

Here, you first checked to make sure that the parser was properly configured with a ContentHandler. (For this app, we don't care about anything else.) You then generated the events for the start of the document and the root element, and finished by sending the end-event for the root element and the end-event for the document.

A couple of items are noteworthy, at this point:

- We haven't bothered to send the `setDocumentLocator` event, since that is optional. Were it important, that event would be sent immediately before the `startDocument` event.
- We've generated an `ignorableWhitespace` event before the end of the root element. This, too, is optional, but it drastically improves readability of the output, as you'll see in a few moments. (In this case, the whitespace consists of a single newline, which is sent the same way that `characters` method are sent: as a character array, a starting index, and a length.)

Now that SAX events are being generated for the document and the root element, the next step is to modify the `output` method to generate the appropriate element events for each data item. Make the changes shown below to do that:

```
void output(String name, String prefix, String line)

throws SAXException
{
    int startIndex = prefix.length() + 2; // 2=length of ": "
    String text = line.substring(startIndex);
    System.out.println(name + ": " + text);

    int textLength = line.length() - startIndex;
    handler.ignorableWhitespace(indent.toCharArray(),
                                0, // start index
                                indent.length()
                                );
    handler.startElement(nsu, name, name /*"qName"*/, atts);
    handler.characters(line.toCharArray(),
                       startIndex,
                       textLength);
    handler.endElement(nsu, name, name);
}
```

Since the ContentHandler methods can send SAXExceptions back to the parser, the parser has to be prepared to deal with them. In this case, we don't expect any, so we'll simply allow the app to fall on its sword and die if any occur.

You then calculate the length of the data, and once again generate some ignorable whitespace for readability. In this case, there is only one level of data, so we can use a fixed indent string. (If the data were more structured, we would have to calculate how much space to indent, depending on the nesting of the data.)

Note:
The indent string makes no difference to the data, but will make the output a lot easier to read. Once everything is working, try generating the result without that string! All of the elements will wind up concatenated end to end, like this:
`<addressbook><nickname>Fred</nickname><email>...`

Next, add the method that configures the parser with the ContentHandler that is to receive the events it generates:

```
/** Allow an application to register a content event handler. */
Public void setContentHandler(ContentHandler handler) {
  this.handler = handler;
}

/** Return the current content handler. */
Public ContentHandler getContentHandler() {
  return this.handler;
}
```

There are several more methods that must be implemented in order to satisfy the XmlReader interface. For the purpose of this exercise, we'll generate null methods for all of them. For a production application, though, you may want to consider implementing the error handler methods to produce a more robust app. For now, though, add the code highlighted below to generate null methods for them:

```
/** Allow an application to register an error event handler. */
Public void setErrorHandler(ErrorHandler handler)
{ }

/** Return the current error handler. */
Public ErrorHandler getErrorHandler()
{ return null; }
```

Finally, add the code highlighted below to generate null methods for the remainder of the XmlReader interface. (Most of them are of value to a real SAX parser, but have little bearing on a data-conversion application like this one.)

```
/** Parse an XML document from a system identifier (URI). */
public void parse(String systemId)
throws IOException, SAXException
{ }

/** Return the current DTD handler. */
Public DTDHandler getDTDHandler()
{ return null; }

/** Return the current entity resolver. */
Public EntityResolver getEntityResolver()
{ return null; }

/** Allow an application to register an entity resolver. */
Public void setEntityResolver(EntityResolver resolver)
{ }

/** Allow an application to register a DTD event handler. */
Public void setDTDHandler(DTDHandler handler)
{ }
```

```
/** Look up the value of a property. */
Public Object getProperty(java.lang.String name)
{ return null; }

/** Set the value of a property. */
Public void setProperty(java.lang.String name, java.lang.Object value)
{ }

/** Set the state of a feature. */
Public void setFeature(java.lang.String name, boolean value)
{ }

/** Look up the value of a feature. */
Public boolean getFeature(java.lang.String name)
{ return false; }
```

Congratulations! You now have a parser you can use to generate SAX events. In the next section, you'll use it to construct a SAX source object that will let you transform the data into XML.

Using the Parser as a SAXSource

Given a SAX parser to use as an event source, you can (quite easily!) construct a transformer to produce a result. In this section, you'll modify the TransformerApp you've been working with to produce a stream output result, although you could just as easily produce a DOM result.

> **Note:**
> The code discussed in this section is in TransformationApp04.java. The results of running it are in TransformationLog04.

> **Important!**
> Be sure to shift gears! Put the AddressBookReader aside and open up the TransformationApp. The work you do in this section affects the TransformationApp!

Start by making the changes shown below to import the classes you'll need to construct a SAXSource object. (You won't be needing the DOM classes at this point, so they are discarded here, although leaving them in doesn't do any harm.)

```
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.ContentHandler;
import org.xml.sax.InputSource;
~~import org.w3c.dom.Document;~~
~~import org.w3c.dom.DOMException;~~
...
~~import javax.xml.transform.dom.DOMSource;~~
import javax.xml.transform.sax.SAXSource;
import javax.xml.transform.stream.StreamResult;
```

Next, remove a few other holdovers from our DOM-processing days, and add the code to create an instance of the AddressBookReader:

```
public class TransformationApp
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

    public static void main(String argv[])
    {
        ...
        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        //factory.setNamespaceAware(true);
        //factory.setValidating(true);

        // Create the sax "parser".
        AddressBookReader saxReader = new AddressBookReader();

       try {
            File f = new File(argv[0]);
            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(f);
```

Guess what! You're almost done. Just a couple of steps to go. Add the code highlighted below to construct a SAXSource object:

```
// Use a Transformer for output
...
Transformer transformer = tFactory.newTransformer();

// Use the parser as a SAX source for input
FileReader fr = new FileReader(f);
BufferedReader br = new BufferedReader(Fr);
InputSource inputSource = new InputSource(Fr);
SAXSource source = new SAXSource(saxReader, inputSource);
StreamResult result = new StreamResult(System.out);
transformer.transform(source, result);
```

Here, you constructed a buffered reader (as mentioned earlier) and encapsulated it in an input source object. You then created a SAXSource object, passing it the reader and the InputSource object, and passed that to the transformer.

When the app runs, the transformer will configure itself as the ContentHandler for the SAX parser (the AddressBookReader and tell the parser to operate on the `inputSource` object. Events generated by the parser will then go to the transformer, which will do the appropriate thing and pass the data on to the result object.

Finally, remove the exceptions you no longer need to worry about, since the TransformationApp no longer generates them:

```
} catch (SAXException sxe) {
   // Error generated by this application
   // (or a parser-initialization error)
   Exception  x = sxe;
   if (sxe.getException() != null)
       x = sxe.getException();
   x.printStackTrace();

} catch (ParserConfigurationException pce) {
    // Parser with specified options can't be built
    pce.printStackTrace();

} catch (IOException ioe) {
```

You're done! You have no created a transformer which will use a SAXSource as input, and produce a StreamResult as output..

Doing the Conversion

Now run the app on the address book file. Your output should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<addressbook>
    <nickname>Fred</nickname>
    <email>fred@barneys.house</email>
    <html>TRUE</html>
    <firstname>Fred</firstname>
    <lastname>Flintstone</lastname>
    <work>999-Quarry</work>
    <home>999-BedrockLane</home>
    <fax>888-Squawk</fax>
    <pager>777-pager</pager>
    <cell>555-cell</cell>
</addressbook>
```

You have now successfully converted an existing data structure to XML . And it wasn't even that hard. Congratulations!

4. Transforming XML Data with XSLT

The XML Stylesheet Language for Transformations (XSLT) can be used for many purposes. For example, you could generate PDF or postscript from the XML data. But generally, XSLT is used to generated formatted HTML output, or to create an alternative XML representation of the data.

In this section of the tutorial, you'll use an XSLT transform to translate XML input data to HTML output.

> **Note:**
> The XSLT specification is very large and quite complex. Rather thick books have been written on the subject. So this tutorial can only scratch the surface. It will give you enough a background to get started, so you can undertake simple XSLT processing tasks. It should also give you a head start when you investigate XSLT further.

Link Summary

Local Links

- XPath addressing

Exercise Links

- TransformationApp02
- Stylizer.java

- article1.xml / article1-xml.html
- article1a.xsl / article1a-xsl.html
- stylizer1a.txt / stylizer1a.html
- article1b.xsl / article1b-xsl.html
- stylizer1b.txt / stylizer1b.html
- article1c.xsl / article1c-xsl.html
- stylizer1c.txt / stylizer1c.html

- article2.xml / article2-xml.html
- article2.xsl / article2-xsl.html
- stylizer2.txt / stylizer2.html

- article3.xml / article3-xml.html
- article3.xsl / article3-xsl.html
- stylizer3.txt / stylizer3.html

API Links

- Transformer

External Links

- Schematron validator
- XSLT Specification
- Java Printing Service (JPS)

Defining an Ultra-Simple `article` Document Type

We'll start by defining a super simple document type that could be used for writing articles. Our `<article>` documents will contain these structure tags:

- `<TITLE>` -- The title of the article.
- `<SECT>` -- A section. (Consists of a *heading* and a *body*.)
- `<PARA>` -- A paragraph.
- `<LIST>` -- A list.
- `<ITEM>` -- An entry in a list.
- `<NOTE>` -- An aside, which will be offset from the main text.

The slightly unusual aspect of this structure is that we won't create a separate element tag for a section heading. Such elements are commonly created to distinguish the heading text (and any tags it contains) from the body of the section (that is, any structure elements underneath the heading).

Instead, we'll allow the heading to merge seamlessly into the body of a section. That arrangement adds some complexity to the stylesheet, but that will give us a chance to explore XSLT's template-selection mechanisms. It also matches our intuitive expectations about document structure, where the text of a heading is directly followed by structure elements, which can simplify outline-oriented editing.

Glossary Terms

DOM, mixed-content model, well-formed

Note:
However, that structure is not easily validated, because XML's mixed-content model allows text anywhere in a section, whereas we want to confine text and inline elements so that they only appear *before* the first structure element in the body of the section. The assertion-based validator (Schematron) can do it, but most other schema mechanisms can't. So we'll dispense with defining a DTD for the document type.

In this structure, sections can be nested. The depth of the nesting will determine what kind of HTML formatting to use for the section heading (for example, h1 or h2.) That's also useful with outline-oriented editing, because it lets you can move sections around at will without having to worry about changing the heading tag -- or any of the other section headings that are affected by the move.

For lists, we'll use a `type` attribute to specify whether the list entries are `unordered` (bulleted), `alpha` (enumerated with lower case letters), `ALPHA` (enumerated with uppercase letters, or `numbered`.

We'll also allow for some inline tags that change the appearance of the text:

- `<B>` -- bold
- `<I>` -- italics
- `<U>` -- underline
- `<DEF>` -- definition
- `<LINK>` -- link to a URL

Note:
An *inline* tag does not generate a line break, so a style change caused by an inline tag does not affect the flow of text on the page (although it will affect the appearance of that text). A *structure* tag, on the other hand, demarcates a new segment of text, so at a minimum it always generates a line break, in addition to other format changes.

The `<DEF>` tag will help make things interesting. That tag will used for terms that are defined in the text. Such terms will be displayed in italics, they way the ordinarily are in a document. But using a special tag in the XML will allow an index to program to one day find such definitions and add them to the index, along with keywords in headings. In the *Note* above, for example, the definitions of inline tags and structure tags could have been marked with `<DEF>` tags, for future indexing.

Finally, the `LINK` tag serves two purposes. First, it will let us create a link to a URL without having to put the URL in twice -- so we can code `<link>http//...</link>` instead of `<a href="http//...">http//...</a>`. Of course, we'll also want to allow a form that looks like `<link target="...">...name...</link>`. That leads to the second reason for the `<link>` tag -- it will give us an opportunity to play with conditional expressions in XSLT.

Note:
As one college professor said, the trick to defining a research project is to find something that is "large enough to be feasible... but small enough to be feasible". Although the article structure is exceedingly simple (consisting of only 11 tags), it raises enough interesting problems to keep us busy exploring XSLT

for a while! Along the way, we'll get a good view of it's basic capabilities. But there will still be large areas of the spec that are left untouched. The last part of this tutorial will point out the major things we missed, to give you some sense of what sorts of features await you in the specification!

Creating a Test Document

Here, you'll create a simple test document using nested `<SECT>` elements, a few `<PARA>` elements, a `<NOTE>` element, a `<LINK>`, and a `<LIST type="unordered">`. The idea is to create a document with one of everything, so we can explore the more interesting translation mechanisms.

> **Note:**
> The sample data described here is contained in article1.xml. (The browsable version is article1-xml.html.)

To make the test document, create a file called **article.xml** and enter the XML data shown below.

```
<?xml version="1.0"?>
<ARTICLE>
  <TITLE>A Sample Article</TITLE>
  <SECT>The First Major Section
     <PARA>This section will introduce a subsection.</PARA>
     <SECT>The Subsection Heading
       <PARA>This is the text of the subsection.
       </PARA>
     </SECT>
  </SECT>
</ARTICLE>
```

Note that in the XML file, the subsection is totally contained within the major section. (Unlike HTML, for example, where headings, do no *contain* the body of a section.) The result is an outline structure that is harder to edit in plain-text form, like this. But much easier to edit with an outline-oriented editor.

Someday, given an tree-oriented XML editor that understands inline tags like `<B>` and `<I>`, it should be possible to edit an article of this kind in outline form, without requiring a complicated stylesheet. (Thereby allowing the writer to focus on the structure of the article, leaving layout until much later in the process.) In such an editor, the article-fragment above would look something like this:

- **`<ARTICLE>`**
  - **`<TITLE>`**A Sample Article
  - **`<SECT>`**The First Major Section
    - **`<PARA>`**This section will introduce a subsection.
    - **`<SECT>`**The Subheading
      - **`<PARA>`**This is the text of the subsection. Note that ...

At the moment, tree-structured editors exist, but they treat inline tags like `<B>` and `<I>` the same way that they treat other structure tags, which can make the "outline" a bit difficult to read. But hopefully, that situation will improve one day. Meanwhile, we'll press on...

Writing an XSLT Transform

In this part of the tutorial, you'll begin writing an XSLT transform that will convert the XML article and render it in HTML.

> **Note:**
> The transform described in this section is contained in article1a.xsl. (The browsable version is article1a-xsl.html.)

Start by creating a normal XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Then add the lines shown below to create an XSL stylesheet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >

</xsl:stylesheet>
```

Now, set it up to produce HTML-compatible output:

```
<xsl:stylesheet
  ...
  >
  <xsl:output method="html"/>

  ...

</xsl:stylesheet>
```

We'll get into the detailed reasons for that entry later on in this section. But for now, note that if you want to output anything besides well-formed XML, then you'll need an `<xsl:output>` tag like the one shown, specifying either `"text"` or `"html"`. (The default value is `"xml"`.)

> **Note:**
> When you specify XML output, you can add the `indent` attribute to produce nicely indented XML output. The specification looks like this: `<xsl:output method="xml" indent="yes"/>`.

Processing the Basic Structure Elements

You'll start filling in the stylesheet by processing the elements that go into creating a table of contents -- the root element, the title element, and headings. You'll also process the PARA element defined in the test document.

> **Note:**
> If on first reading you skipped the section of this tutorial that discusses the XPath addressing mechanisms,

now is a good time to go back and review that section!

Begin by adding the main instruction that processes the root element:

```
<xsl:stylesheet ...
   <xsl:template match="/">
      <html><body>
        <xsl:apply-templates/>
      </body></html>
   </xsl:template>

</xsl:stylesheet>
```

The XSL commands are shown in bold. (Note that they are defined in the "xsl" namespace.) The instruction `<xsl:apply-templates>` processes the children of the current node. In the case, the current node is the root node.

Despite its simplicity,. this example illustrates a number of important ideas, so it's worth understanding thoroughly. The first concept is that a stylesheet contains a number of *templates*, defined with the <xsl:template> tag. Each template contains a `match` attribute, which selects the elements that the template will be applied to, using the XPath addressing mechanisms.

Within the template, tags that do not start with the `xsl:` namespace prefix are simply copied. The newlines and whitespace that follow them are also copied, which helps to format make the resulting output readable.

> **Note:**
> When a newline is not present, whitespace generally seems to be ignored. To include whitespace in the output in such cases, or to include other text, you can use the `<xsl:text>` tag. Basically, an XSLT stylesheet expects to process tags. So everything it sees needs to be either an `<xsl:..>` tag, some other tag, or whitespace.

In this case, the non-xsl tags are HTML tags (shown in red, for readability). So when the root tag is matched, XSLT outputs the HTML start-tags, processes any templates that apply to children of the root, and then outputs the HTML end-tags.

Process the <TITLE> element

Next, add a template to process the article title:

```
   <xsl:template match="/ARTICLE/TITLE">
      <h1 align="center"> <xsl:apply-templates/> </h1>
   </xsl:template>

</xsl:stylesheet>
```

In this case, you specified a complete path to the TITLE element, and output some HTML to make the text of the title into a large, centered heading. In this case, the apply-templates tag ensures that if the title contains any inline tags like italics, links, or underlining, they will be processed as well.

More importantly, the apply-templates instruction causes the *text* of the title to be processed. Like the DOM data model, the XSLT data model is based on the concept of *text nodes* hanging off of *element nodes* (which, in turn, can hang off other element nodes, and so on). That hierarchical structure constitutes the source tree. There is also a result tree, which contains the output.

XSLT works by transforming the source tree into the result tree. To visualize the result of XSLT operations, it is helpful to understand the structure of those trees, and their contents. (For more on this subject, see the sidebar on the XSLT Data Model later in this section.)

Process headings

To continue processing the basic structure elements, add a template to process the top-level headings:

```
    <xsl:template match="/ARTICLE/SECT">
       <h1> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </h1>
       <xsl:apply-templates select="SECT|PARA|LIST|NOTE"/>
    </xsl:template>

 </xsl:stylesheet>
```

Here, you've specified the path to the topmost `SECT` elements. But this time, you've applied templates in two stages, using the `select` attribute. For the first stage, you selected text nodes using the XPath `text()` function, as well as inline tags like bold and italics. (The vertical pipe (|) is used to match multiple items -- text, *or* a bold tag, *or* an italics tag, etc.) In the second stage, you selected the other structure elements contained in the file, for sections, paragraphs, lists, and notes.

Using the select tags let you put the text and inline elements between the `<h1>...</h1>` tags, while making sure that all of the structure tags in the section are processed afterwards. In other words, you made sure that the nesting of the headings in the XML document is *not* reflected in the HTML formatting, which is important for HTML output.

In general, the select clause lets you apply all templates to a selected subset of the information available at the current context. As another example, this template selects all attributes of the current node:

```
 <xsl:apply-templates select="@*"/></attributes>
```

Next, add the virtually identical template to process the second-level headings:

```
    <xsl:template match="/ARTICLE/SECT/SECT">
       <h2> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </h2>
       <xsl:apply-templates select="SECT|PARA|LIST|NOTE"/>
    </xsl:template>

 </xsl:stylesheet>
```

Generate a runtime message

You could add templates for deeper headings, too, but at some point you have to stop, if only because HTML only goes down to 5 levels. But for this example, you'll stop at two levels of section headings. But if the XML input happens to contain a 3rd level, you'll want to deliver an error message to the user. This section shows you how to do that.

> **Note:**
> We *could* continue processing SECT elements that are further down, by selecting them with the expression `/SECT/SECT//SECT`. The `//` selects any SECT elements, at any depth", as defined by XPath addressing mechanism. But we'll take the opportunity to play with messaging, instead.

Add the following template to generate an error when a section is encountered that is nested too deep:

```
  <xsl:template match="/ARTICLE/SECT/SECT/SECT">
    <xsl:message terminate="yes">Error: Sections can only be nested 2 
deep.</xsl:message>
  </xsl:template>

</xsl:stylesheet>
```

The `terminate="yes"` clause causes the transformation process to stop after the message is generated. Without it, processing could still go on with everything in that section being ignored.

> **Extra-Credit Exercise:**
> Expand the stylesheet to handle sections nested up to 5 sections deep, generating <h1>..<h5> tags. Generate an error on any section nested 6 levels deep.

Finally, finish up the stylesheet by adding a template to process the PARA tag:

```
  <xsl:template match="PARA">
    <p><xsl:apply-templates/></p>
  </xsl:template>

</xsl:stylesheet>
```

Nothing unusual here. Just another template like the ones you're used to.

Writing the Basic Program

In this part of the tutorial, you'll modify the program that used XSLT to echo an XML file unchanged, and modify it so that it uses your stylesheet.

> **Note:**
> The code shown in this section is contained in Stylizer.java. The result is the HTML code shown in stylizer1a.txt. (The displayable version is stylizer1a.html.)

Start by copying TransformationApp02, which parses an XML file and writes to System.out. Save it as **Stylizer.java**.

Next, modify occurrences of the class name and the usage-section of the program:

```
public class ~~TransformationApp~~Stylizer
{
    if (argv.length != ~~1~~ 2) {
        System.err.println ("Usage: java ~~TransformationApp~~Stylizer stylesheet
filename");
        System.exit (1);
    }
    ...
```

Then modify the program to use the stylesheet when creating the Transformer object.

```
...
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;
...

public class Stylizer
{
  ...
  public static void main (String argv[])
  {
    ...
    try {
      ~~File f = new File(arv[0]);~~
      File stylesheet = new File(argv[0]);
      File datafile   = new File(argv[1]);

      DocumentBuilder builder = factory.newDocumentBuilder();
      document = builder.parse(~~f~~ datafile);
      ...
      StreamSource stylesource = new StreamSource(stylesheet);
      Transformer transformer = tFactory.newTransformer(stylesource);
      ...
```

This code uses the file to create a StreamSource object, and then passes the source object to the factory class to get the transformer.

Note:
You can simplify the code somewhat by eliminating the DOMSource class entirely. Instead of creating a DOMSource object for the XML file, create a StreamSource object for it, as well as for the stylesheet. (Take it on for extra credit!)

Now compile and run the program using article1a.xsl on article1.xml. The results should look like this:

```
<html>
<body>

<h1 align="center">A Sample Article</h1>

<h1>The First Major Section


  </h1>
```

```
<p>This section will introduce a subsection.</p>
<h2>The Subsection Heading

     </h2>
<p>This is the text of the subsection.
       </p>

</body>
</html>
```

At this point, there is quite a bit of excess whitespace in the output. You'll see how to eliminate most of it in the next section.

Trimming the Whitespace

If you recall, when you took a look at the structure of a DOM, there were many text nodes that contained nothing but ignorable whitespace. Most of the excess whitespace in the output came from them. Fortunately, XSL gives you a way to eliminate them. (For more about the node structure, see the sidebar: *The XSLT/XPath Data Model*.)

> **Note:**
> The stylesheet described here is article1b.xsl. The result is the HTML code shown in stylizer1b.txt. (The displayable versions are article1b-xsl.html and stylizer1b.html.)

To do remove some of the excess whitespace, add the line highlighted below to the stylesheet.

The XSLT/XPath Data Model

Like the DOM, the XSL/XPath data model consists of a tree containing a variety of nodes. Under any given element node, there are text nodes, attribute nodes, element nodes, comment nodes, and processing instruction nodes.

Once an XPath expression establishes a *context*, other expressions produce values that are relative to that context. For example, the expression `//LIST` establishes a context consisting of a LIST node. Within the XSLT template that processes such nodes, the expression `@type` refers to the element's type attribute. (Similarly, the expression `@*` refers to all of the element's attributes.)

```
<xsl:stylesheet ...
  >
  <xsl:output method="html"/>
  <xsl:strip-space elements="SECT"/>
  ...
```

This instruction tells XSL to remove any text nodes under `SECT` elements that contain nothing but whitespace. Nodes that contain text other than whitespace will not be affected, and other kinds of nodes are not affected.

Now, when you run the program, the result looks like this:

```
<html>
<body>

<h1 align="center">A Sample Article</h1>

<h1>The First Major Section
     </h1>
```

```
<p>This section will introduce a subsection.</p>
<h2>The Subsection Heading
       </h2>
<p>This is the text of the subsection.
       </p>

</body>
</html>
```

That's quite an improvement. There are still newline characters and white space after the headings, but those come from the way the XML is written:

```
<SECT>The First Major Section
____<PARA>This section will introduce a subsection.</PARA>
^^^^
```

Here, you can see that the section heading ends with a newline and indentation space, before the PARA entry starts. That's not a big worry, because the browsers that will process the HTML routinely compress and ignore the excess space. But we there is still one more formatting at our disposal.

Note:
The stylesheet described here is article1c.xsl. The result is the HTML code shown in stylizer1c.txt. (The displayable versions are article1c-xsl.html and stylizer1c.html.)

To get rid of that last little bit of whitespace, add this template to the stylesheet:

```
  <xsl:template match="text()">
    <xsl:value-of select="normalize-space()"/>
  </xsl:template>

</xsl:stylesheet>
```

The output now looks like this:

```
<html>
<body>
<h1 align="center">A Sample Article</h1>
<h1>The First Major Section</h1>
<p>This section will introduce a subsection.</p>
<h2>The Subsection Heading</h2>
<p>This is the text of the subsection.</p>
</body>
</html>
```

That is quite a bit better. Of course, it would be nicer if it were indented, but that turns out to be somewhat harder than expected! Here are some possible avenues of attack, along with the difficulties:

- **Indent option:** Unfortunately, the `indent="yes"` option that can be applied to XML output is not available for HTML output. Even if that option were available, it wouldn't help, because HTML elements are rarely nested!

Although HTML source is frequently indented to show the *implied* structure, the HTML tags themselves are not nested in a way that creates a *real* structure.

- **Indent variables:** The `<xsl:text>` function lets you add any text you want, including whitespace. So, it could conceivably be used to output indentation space. The problem is to vary the *amount* of indentation space. XSLT variables seem like a good idea, but they don't work here. The reason is that when you assign a value to a variable in a template, the value is only known *within* that template (statically, at compile time value). Even if the variable is defined globally, the assigned value is not stored in a way that lets it be dynamically known by other templates at runtime. Once `<apply-templates/>` invokes other templates, they are unaware of any variable settings made in other templates.

- **Parameterized templates:** Using a "parameterized template" is another way to modify a template's behavior. But determining the amount of indentation space to pass as the parameter remains the crux of the problem!

At the moment, then, there does not appear to be any good way to control the indentation of HTML-formatted output. Typically, that fact is of little consequence, since the data will usually be manipulated in its XML form, while the HTML version is only used for display a browser. It's only inconvenient in a tutorial like this, where it would be nice to see the structure you're creating! But when you click on the link to stylizer1c.html, you see the results you expect.

Processing the Remaining Structure Elements

In this section, you'll process the LIST and NOTE elements that add additional structure to an article.

> **Note:**
> The sample document described in this section is article2.xml, the stylesheet used to manipulate it is article2.xsl. The result is the HTML code shown in stylizer2.txt. (The displayable versions are article2-xml.html, article2-xsl.html, and stylizer2.html.)

Start by adding some test data to the sample document:

```
<?xml version="1.0"?>
<ARTICLE>
  <TITLE>A Sample Article</TITLE>
  <SECT>The First Major Section
    ...
  </SECT>
  <SECT>The Second Major Section
     <PARA>This section adds a LIST and a NOTE.
     <PARA>Here is the LIST:
       <LIST type="ordered">
           <ITEM>Pears</ITEM>
           <ITEM>Grapes</ITEM>
       </LIST>
     </PARA>
     <PARA>And here is the NOTE:
       <NOTE>Don't forget to go to the hardware store on your
             way to the grocery!
       </NOTE>
     </PARA>
  </SECT>
</ARTICLE>
```

Note:
Although the list and note in the XML file are contained in their respective paragraphs, it really makes no difference whether they are contained or not -- the generated HTML will be the same, either way. But having them contained will make them easier to deal with in an outline-oriented editor.

Modify <PARA> handling

Next, modify the PARA template to account for the fact that we are now allowing some of the structure elements to be embedded with a paragraph:

```
<xsl:template match="PARA">
  <p><xsl:apply-templates/></p>
  <p> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </p>
  <xsl:apply-templates select="PARA|LIST|NOTE"/>
</xsl:template>
```

This modification uses the same technique you used for section headings. The only difference is that SECT elements are not expected within a paragraph.

Process <LIST> and <ITEM> elements

Now you're ready to add a template to process LIST elements:

```
  <xsl:template match="LIST">
    <xsl:if test="@type='ordered'">
      <ol>
        <xsl:apply-templates/>
      </ol>
    </xsl:if>
    <xsl:if test="@type='unordered'">
      <ul>
        <xsl:apply-templates/>
      </UL>
    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

The `<xsl:if>` tag uses the `test=""` attribute to specify a boolean condition. In this case, the value of the `type` attribute is tested, and the list that is generated changes depending on whether the value is `ordered` or `unordered`.

The two important things to note for this example are:

1. There is no `else` clause, nor is there a `return` or `exit` statement, so it takes two `<xsl:if>` tags to cover the two options. (Or the `<xsl:choose>` tag could have been used, which provides case-statement functionality.)

2. Single quotes are required around the attribute values. Otherwise, the XSLT processor attempts to interpret the word `ordered` as an XPath function, instead of as a string

Now finish up `LIST` processing by handling `ITEM` elements. Nothing spectacular here.

```
    <xsl:template match="ITEM">
      <li><xsl:apply-templates/>
      </li>
    </xsl:template>

  </xsl:stylesheet>
```

Ordering Templates in a Stylesheet

By now, you should have the idea that templates are independent of one another, so it doesn't generally matter where they occur in a file. So from here on, we'll just show the template you need to add. (For the sake of comparison, they're always added at the end of the example stylesheet.)

Order *does* make a difference when two templates can apply to the same node, In that case, the one that is defined *last* is the one that is found and processed. For example, to change the ordering of an indented list to use lowercase alphabetics, you could specify a template pattern that looks like this: `//LIST//LIST`. In that template, you would use the HTML option to generate an alphabetic enumeration, instead of a numeric one.

But such an element could also be identified by the pattern `//LIST`. To make sure the proper processing is done, the template that specifies `//LIST` would have to appear *before* the template the specifies `//LIST//LIST`.

Process <NOTE> elements

The last remaining structure element is the `NOTE` element. Add the template shown below to handle that.

```
  <xsl:template match="NOTE">
    <blockquote><p><b>Note:</b><br/>
      <xsl:apply-templates/>
    </p></blockquote>
  </xsl:template>
```

This code brings up an interesting issue that results from the inclusion of the `<Br/>` tag. To be well-formed XML, the tag must be specified in the stylesheet as `<Br/>`, but that tag is not recognized by many browsers. And while most browsers recognize the sequence `<Br></Br>`, they all treat it like a paragraph break, instead of a single line break.

In other words, the transformation *must* generate a `<Br>` tag, but the stylesheet must specify `<Br/>`. That brings us to the major reason for that special output tag we added early in the stylesheet:

```
  <xsl:stylesheet ... >
    <xsl:output method="html"/>
    ...
  </xsl:stylesheet>
```

That output specification converts empty tags like `<Br/>` to their HTML form, `<Br>,` on output. That conversion is important, because most browsers do not recognize the empty-tags. Here is a list of the affected tags:

• `area` • `base` • `basefont` • `Br` • `col`	• `frame` • hr • img • input	• `isindex` • `link` • `meta` • `param`

Summarizing:
By default, XSLT produces well-formed XML on output. And since an XSL stylesheet is well-formed XML to start with, you cannot easily put a tag like `<Br>` in the middle of it. The "`<xsl:output method="html"/>`" solves the problem, so you can code `<Br/>` in the stylesheet, but get `<Br>` in the output.

The other major reason for specifying `<xsl:output method="html"/>` is that, like the specification `<xsl:output method="text"/>`, generated text is *not* escaped. For example, if the stylesheet includes the `&lt;` entity reference, it will appear as the "<" character in the generated text. When XML is generated, on the other hand, the `&lt;` entity reference in the stylesheet would be unchanged, so it would appear as `&lt;` in the generated text.

Note:
If you actually want &lt; to be generated as part of the HTML output, you'll need to encode it as `&amp;lt;` -- that sequence becomes &lt; on output, because only the `&amp;` is converted to an & character.

Run the program

Here is the HTML that is generated for the second section when you run the program now:

```
...
<h1>The Second Major Section</h1>
<p>This section adds a LIST and a NOTE.</p>
<p>Here is the LIST:</p>
<ol>
<li>Pears</li>
<li>Grapes</li>
</ol>
<p>And here is the NOTE:</p>
<blockquote>
<b>Note:</b>
<Br>Don't forget to go to the hardware store on your way to the grocery!
</blockquote>
```

Process Inline (Content) Elements

The only remaining tags in the `ARTICLE` type are the *inline* tags -- the ones that don't create a line break in the output, but which instead are integrated into the stream of text they are part of.

Inline elements are different from structure elements, in that they are part of the content of a tag. If you think of an element as a node in a document tree, then each node has both *content* and *structure*. The content is composed of the text and inline tags it contains. The structure consists of the other elements (structure elements) under the tag.

> **Note:**
> The sample document described in this section is article3.xml, the stylesheet used to manipulate it is article3.xsl. The result is the HTML code shown in stylizer3.txt. (The browser-displayable versions are article3-xml.html, article3-xsl.html, and stylizer3.html.)

Start by adding one more bit of test data to the sample document:

```
<?xml version="1.0"?>
<ARTICLE>
  <TITLE>A Sample Article</TITLE>
  <SECT>The First Major Section
    ...
  </SECT>
  <SECT>The Second Major Section
     ...
  </SECT>
  <SECT>The <I>Third</I> Major Section
     <PARA>In addition to the inline tag in the heading, this section
           defines the term <DEF>inline</DEF>, which literally means
           "no line break". It also adds a simple link to the main page
           for the Java platform (<LINK>http://java.sun.com</LINK>),
           as well as a link to the
           <LINK target="http://java.sun.com/xml">XML</LINK> page.
     </PARA>
  </SECT>
</ARTICLE>
```

Now, process the inline `<DEF>` elements in paragraphs, renaming them to HTML italics tags:

```
<xsl:template match="DEF">
   <i> <xsl:apply-templates/> </i>
</xsl:template>
```

Next, comment out the text-node normalization. It has served its purpose, and new we're to the point that we need to preserve spaces important:

```
<!--
  <xsl:template match="text()">
    <xsl:value-of select="normalize-space()"/>
  </xsl:template>
-->
```

This modification keeps us from losing spaces before tags like `<I>` and `<DEF>`. (Try the program without this modification to see the result.)

Now, process basic inline HTML elements like <B>, <I>, <U> for bold, italics, and underlining.

```
<xsl:template match="B|I|U">
  <xsl:element name="{name()}">
     <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

The `<xsl:element>` tag lets you compute the element you want to generate. Here, you generate the appropriate the inline tag using the name of the current element. In particular, note the use of curly braces (`{}`) in the `name=".."` expression. Those curly braces cause the text inside the quotes to be processed as an XPath expression, instead of being interpreted as a literal string. Here, they cause the XPath `name()` function to return the name of the current node.

Curly braces are recognized anywhere that an "attribute value template" can occur. (Attribute value templates are defined in section 7.6.2 of the specification, and they appear several places in the template definitions.). In such expressions, curly braces can also be used to refer to the value of an attribute, `{@foo}`, or to the content of an element `{foo}`.

> **Note:**
> You can also generate attributes using `<xsl:attribute>`. For more information see Section 7.1.3 of the XSLT Specification.

The last remaining element is the `LINK` tag. The easiest way to process that tag will be to set up a named-template that we can drive with a parameter:

```
<xsl:template name="htmLink">
  <xsl:param name="dest" select="UNDEFINED"/>
  <xsl:element name="a">
    <xsl:attribute name="href">
      <xsl:value-of select="$dest"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

The major difference in this template is that, instead of specifying a `match` clause, you gave the template a name with the `name`="" clause. So this template only gets executed when you invoke it.

Within the template, you also specified a parameter named "dest", using the `<xsl:param>` tag. For a bit of error checking, you used the `select` clause to give that parameter a default value of "`UNDEFINED`". To reference the variable in the `<xsl:value-of>` tag, you specified `"$dest"`.

> **Note:**
> Recall that an entry in quotes is interpreted as an expression, unless it is further enclosed in single quotes. That's why the single quotes were needed earlier, in `"@type='ordered'"` -- to make sure that `ordered` was interpreted as a string.

The `<xsl:element>` tag generates an element. Previously, we have been able to simply specify the element we want by coding something like `<html>`. But here you are dynamically generating the content of the HTML anchor (`<a>`) in the body of the `<xsl:element>` tag. And you are dynamically generating the `href` attribute of the anchor using the `<xsl:attribute>` tag.

The last important part of the template is the `<apply-templates>` tag, which inserts the text from the text node under the `LINK` element. (Without it, there would be no text in the generated HTML link.)

Next, add the template for the `LINK` tag, and call the named template from within it:

```
<xsl:template match="LINK">
  <xsl:if test="@target">
    <!--Target attribute specified.-->
    <xsl:call-template name="htmLink">
      <xsl:with-param name="dest" select="@target"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>

<xsl:template name="htmLink">
  ...
```

The `test="@target"` clause returns true if the `target` attribute exists in the `LINK` tag. So this if-statement generates HTML links when the text of the link and the target defined for it are different.

The `<xsl:call-template>` tag invokes the named template, while `<xsl:with-param>` specifies a parameter using the `name` clause, and its value using the `select` clause.

As the very last step in the stylesheet construction process, add the if-clause shown below to process `LINK` tags that do not have a `target` attribute.

```
<xsl:template match="LINK">
  <xsl:if test="@target">
    ...
  </xsl:if>

  <xsl:if test="not(@target)">
    <xsl:call-template name="htmLink">
      <xsl:with-param name="dest">
        <xsl:apply-templates/>
      </xsl:with-param>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

The `not(...)` clause inverts the previous test (there is no else clause, remember?). So this part of the template is interpreted when the `target` attribute is not specified. This

The Trouble with Variables

It is awfully tempting to create a single template and set a variable for the destination of the link, rather than going to the trouble of setting up a parameterized template and calling it two different ways. The idea would be to set the variable to a default value (say, the text of the `LINK` tag) and then, if `target` attribute exists, set the destination variable to the value of the `target` attribute.

That would be a darn good idea -- if it worked. But once again, the issue is that variables are only known in the scope within which they are defined. So when you code an `<xsl:if>` to change the value of the variable, the value is only known within the context of the `<xsl:if>` tag. Once `</xsl:if>` is encountered, any change to the variable's setting is lost.

A similarly tempting idea is the possibility of replacing the `text()|B|I|U|DEF|LINK` specification with a variable (`$inline`). But since the value of the variable is time, the parameter value comes not from a select clause, but from the *contents* of the `<xsl:with-param>` element.

determined by where it is defined, the value of a global `inline` variable consists of text nodes, `<B>` nodes, etc. that happen to exist at the root level. In other words, the value of such a variable, in this case, is null.

> **Note:**
> Just to make it explicit: variables (which we'll mention a bit later) and parameters can have their value specified *either* by a `select` clause, which lets you use XPath expressions, *or* by the content of the element, which lets you use XSLT tags.

The content of the parameter, in this case, is generated by the `<xsl:apply-templates/>` tag, which inserts the contents of the text node under the `LINK` element.

Run the program

When you run the program now, the results should look like this:

```
...
<h1>The <I>Third</I> Major Section
     </h1>
<p>In addition to the inline tag in the heading, this section
           defines the term <i>inline</i>, which literally means
           "no line break". It also adds a simple link to the main page
           for the Java platform (<a
href="http://java.sun.com">http://java.sun.com</a>),
           as well as a link to the
           <a href="http://java.sun.com/xml">XML</a> page.
     </p>
```

Awesome! You have now converted a rather complex XML file to HTML. (As seemingly simple as it was, it still provided a lot of opportunity for exploration.)

Printing the HTML

You have now converted an XML file to HTML. One day, someone will produce an HTML-aware printing engine that you'll be able to find and use through the Java Printing Service (JPS) API. At that point, you'll have ability to print an arbitrary XML file as formatted data -- all you'll have to do is set up a stylesheet!

What Else Can XSLT Do?

As lengthy as this section of the tutorial has been, it has still only scratched the surface of XSLT's capabilities. Many additional possibilities await you in the XSLT Specification. Here are a few of the things to look for:

- **import** (Section 2.6.2) and **include** (Section 2.6.1)
  Use these statements to modularize and combine XSLT stylesheets. The `include` statement simply inserts any definitions from the included file. The `import` statement lets you override definitions in the imported file with definitions in your own stylesheet.

- **for-each loops** (Section 8)

Loop over a collection of items and process each one, in turn.

- **choose** (case-statement) for conditional processing (Section 9.2)
  Branch to one of multiple processing paths depending on an input value.

- **generating numbers** (Section 7.7)
  Dynamically generate numbered sections, numbered elements, and numeric literals. XSLT provides three numbering modes:
  single: Numbers items under a single heading, like an "ordered list" in HTML..
  multiple: Produces multi-level numbering like "A.1.3".
  any: Consecutively numbers items wherever they appear, like the footnotes in a chapter.

- **formatting numbers** (Section 12.3)
  Control enumeration formatting, so you get numerics (`format="1"`), uppercase alphabetics (`format="A"`), lowercase alphabetics (`format="a"`), or compound numbers, like "A.1", as well as numbers and currency amounts suited for a specific international locale.

- **sorting output** (Section 10)
  Produce output in some desired sorting order.

- **mode-based templates** (Section 5.7)
  Lets you process an element multiple times, each time in a different "mode". You add a `mode` attribute to templates, and then specify `<apply-templates mode="...">` to apply only the templates with a matching mode. Combined with the `<apply-templates select="...">` to slice and dice the input processing, creating a matrix of elements to process and the templates to apply to them.

- **variables** (Section 11)
  Variables, like parameters, let you control a template's behavior. But they are not as valuable as you might think. The value of a variable is only known within the scope of the current template or <xsl:if> clause (for example) in which it is defined. You can't pass a value from one template to another, or even from an enclosed part of a template to another part of the same template.

  These statements are true even for a "global" variable. You can change its value in a template, but the change only applies to that template. And when the expression used to define the global variable is evaluated, that evaluation takes place in the context of the structure's root node. In other words, global variables are essentially runtime constants. Those constants can be useful to change the behavior of a template, especially when coupled with `include` and `import` statements. But variables are not a general-purpose data-management mechanism.

Next...

The final page of the XSLT tutorial will show you how to concatenate multiple transformations together in a filter chain..

---

5. Concatenating XSLT Transformations with a Filter Chain

It is sometimes useful to create a "filter chain" of XSLT transformations, so that the output of one transformation becomes the input of the next. This section of the tutorial shows you how to do that.

Link Summary

Exercise Links

- FilterChain.java
- small-docbook-article.xml
- small-docbook-article-xml.html
- docbookToArticle.xsl
- docbookToArticle-xsl.html
- article1c.xsl / article1c-xsl.html
- filterout.txt / filterout.html

API Links

- Transformer

External Links

- DocBook article format

Writing the Program

Start by writing a program to do the filtering. This example will show the full source code, but you can use one of the programs you've been working on as a basis, to make things easier.

> **Note:**
> The code described here is contained in FilterChain.java.

The sample program includes the import statements that identify the package locations for each class:

```
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;
import org.xml.sax.XMLFilter;

import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerConfigurationException;

import javax.xml.transform.sax.SAXTransformerFactory;
import javax.xml.transform.sax.SAXSource;
import javax.xml.transform.sax.SAXResult;

import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;

import java.io.*;
```

The program also includes the standard error handlers you're used to. They're listed here, just so they are all gathered together in one place:

```
        }
        catch (TransformerConfigurationException tce) {
          // Error generated by the parser
          System.out.println ("\n** Transformer Factory error");
          System.out.println("   " + tce.getMessage() );

          // Use the contained exception, if any
          Throwable x = tce;
          if (tce.getException() != null)
             x = tce.getException();
          x.printStackTrace();
        }
        catch (TransformerException te) {
          // Error generated by the parser
          System.out.println ("\n** Transformation error");
          System.out.println("   " + te.getMessage() );

          // Use the contained exception, if any
          Throwable x = te;
          if (te.getException() != null)
             x = te.getException();
          x.printStackTrace();
        }
        catch (SAXException sxe) {
          // Error generated by this application
          // (or a parser-initialization error)
          Exception  x = sxe;
          if (sxe.getException() != null)
             x = sxe.getException();
          x.printStackTrace();
        }
        catch (ParserConfigurationException pce) {
          // Parser with specified options can't be built
          pce.printStackTrace();
        }
        catch (IOException ioe) {
          // I/O error
          ioe.printStackTrace();
        }
```

In between the import statements and the error handling, the core of the program consists of the code shown below.

```
    public static void main (String argv[])
    {
      if (argv.length != 3) {
        System.err.println ("Usage: java FilterChain stylesheet1 stylesheet2 xmlfile");
        System.exit (1);
      }

      try {
        // Read the arguments
        File stylesheet1 = new File(argv[0]);
        File stylesheet2 = new File(argv[1]);
        File datafile    = new File(argv[2]);
        // Set up the input stream
        BufferedInputStream bis = new BufferedInputStream(new FileInputStream(datafile));
        InputSource input = new InputSource(bis);

        // Set up to read the input file
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser parser = spf.newSAXParser();
        XMLReader reader = parser.getXMLReader();

        // Create the filters (see Note #1)
        SAXTransformerFactory stf =
          (SAXTransformerFactory) TransformerFactory.newInstance();
        XMLFilter filter1 = stf.newXMLFilter(new StreamSource(stylesheet1));
        XMLFilter filter2 = stf.newXMLFilter(new StreamSource(stylesheet2));

        // Wire the output of the reader to filter1 (see Note #2)
        // and the output of filter1 to filter2
        filter1.setParent(reader);
        filter2.setParent(filter1);

        // Set up the output stream
        StreamResult result = new StreamResult(System.out);

        // Set up the transformer to process the SAX events generated
        // by the last filter in the chain
        Transformer transformer = stf.newTransformer();
        SAXSource transformSource = new SAXSource(filter2, input);
        transformer.transform(transformSource, result);
      } catch (...) {
        ...
```

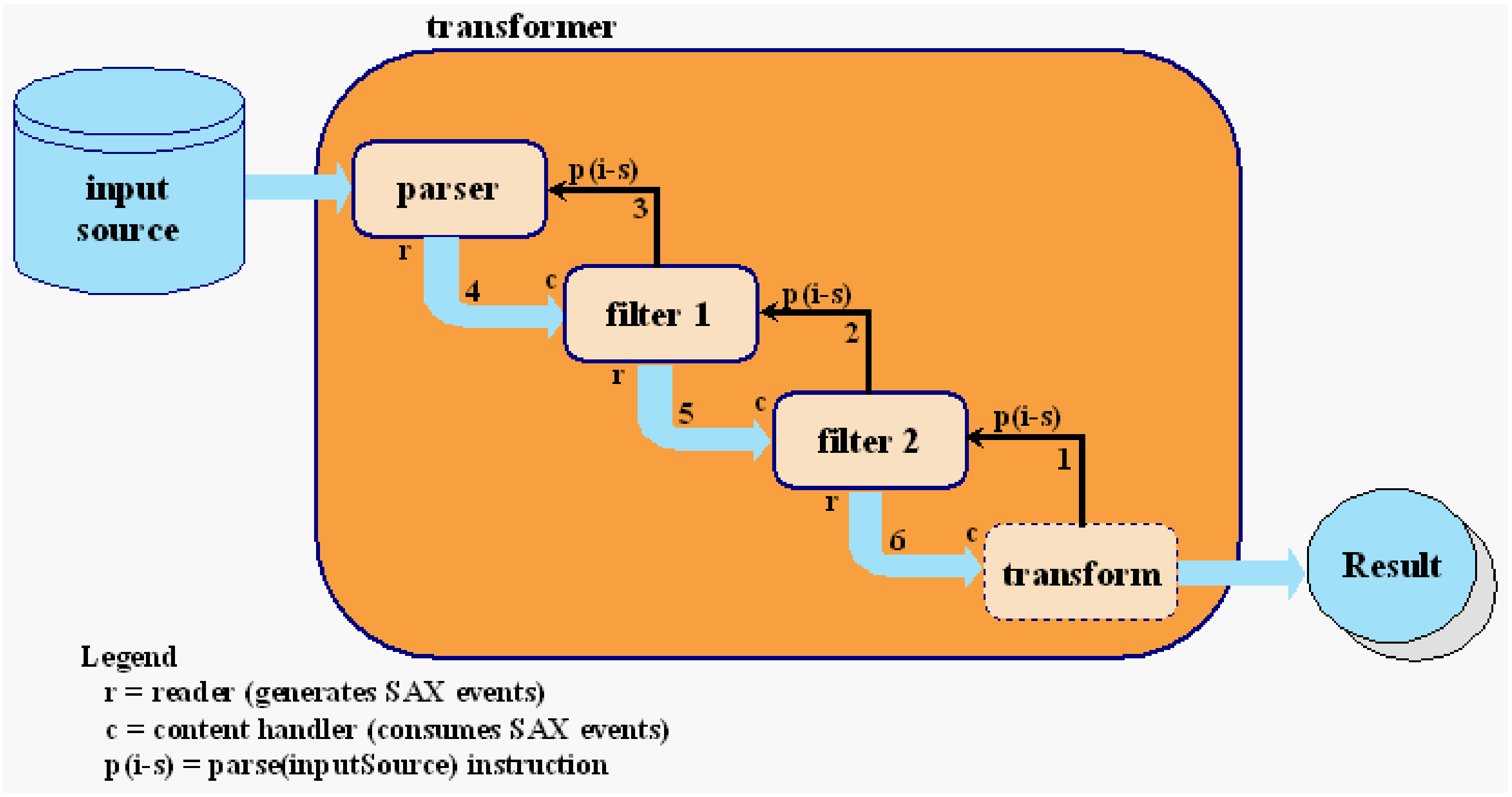
Notes

1. This weird bit of code is explained by the fact that SAXTransformerFactory extends TransformerFactory, adding methods to obtain filter objects. The `newInstance()` method is a static method defined in TransformerFactory, which (naturally enough) returns a TransformerFactory object. In reality, though, it returns a SAXTransformerFactory. So, to get at the extra methods defined by SAXTransformerFactory, the return value must be cast to the actual type.

2. An XMLFilter object is both a SAX reader and a SAX content handler. As a SAX reader, it generates SAX events to whatever object has registered to receive them. As a content handler, it consumes SAX events generated by it's "parent" object -- which is, of necessity, a SAX reader, as well. (Calling the event generator a "parent" must make sense when looking at the internal architecture. From the external perspective, the name doesn't appear to be particularly fitting.) The fact that filters both generate and consume SAX events allows them to be chained together.

Understanding How it Works

The code listed above shows you how to set up the transformation. The diagram below should help you get a better feel for what's happening when it executes.


transformer
input source
parser
p (i-s)
3
r
4
c
filter 1
p (i-s)
2
r
5
c
filter 2
p(i-s)
1
r
6
c
transform
Result
Legend
r = reader (generates SAX events)
c = content handler (consumes SAX events)
p (i-s) = parse(inputSource) instruction

When you create the transformer, you pass it at a `SAXSource` object, which encapsulates a reader (in this case, `filter2`) and an input stream. You also pass it a pointer to the result stream, where it directs its output. The diagram shows what happens when you invoke `transform()` on the transformer. Here is an explanation of the steps:

1. The transformer sets up an internal object as the content handler for `filter2`, and tells it to parse the input source.
2. `filter2`, in turn, sets itself up as the content handler for `filter1`, and tells *it* to parse the input source.
3. Continuing to pass the buck, `filter1` asks the `parser` object to please parse the input source.
4. The `parser` does so, generating SAX events which it passes to `filter1`.
5. `filter1`, acting in its capacity as a content handler, processes the events and does its transformations. Then, acting in its capacity as a SAX reader (XMLReader), it sends SAX events to `filter2`.
6. `filter2` does the same, sending its events to the transformer's content handler, which generates the output stream.

Testing the Program

To try out the program, you'll create an XML file based on a tiny fraction of the XML DocBook format, and convert it to the `ARTICLE` format defined here. Then you'll apply the `ARTICLE` stylesheet to generate an HTML version.

> **Note:**
> This example processes small-docbook-article.xml using docbookToArticle.xsl, and article1c.xsl. The result is the HTML code shown in filterout.txt. (The browser-displayable versions are small-docbook-article-xml.html, docbookToArticle-xsl.html, article1c-xsl.html, and filterout.html.) See the O'Reilly web pages for a good description of the DocBook article format.

Start by creating a small article that uses a minute subset of the XML DocBook format:

```
<?xml version="1.0"?>
<Article>
  <ArtHeader>
    <Title>Title of my (Docbook) article</Title>
  </ArtHeader>
```

```
    <Sect1>
      <Title>Title of Section 1.</Title>
      <Para>This is a paragraph.</Para>
    </Sect1>
</Article>
```

Next, create a stylesheet to convert it into the `ARTICLE` format:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >
  <xsl:output method="xml"/> (see Note #1)

  <xsl:template match="/">
    <ARTICLE>
       <xsl:apply-templates/>
    </ARTICLE>
  </xsl:template>

  <!-- Lower level titles strip out the element tag --> (see Note #2)

  <!-- Top-level title -->
  <xsl:template match="/Article/ArtHeader/Title"> (see Note #3)
    <TITLE> <xsl:apply-templates/> </TITLE>
  </xsl:template>

  <xsl:template match="//Sect1"> (see Note #4)
      <SECT><xsl:apply-templates/></SECT>
  </xsl:template>

  <xsl:template match="Para">
      <PARA><xsl:apply-templates/></PARA> (see Note #5)
  </xsl:template>

</xsl:stylesheet>
```

Notes:

1. This time, the stylesheet is generating XML output.
2. The element below matches the main title. For section titles, the tag gets stripped. (Since no template conversion governs those title elements, they are ignored. The text nodes they contain, however, are still echoed as a result of XSLT's built in template rules (more on that below).
3. The title from the DocBook article header becomes the `ARTICLE` title.
4. Numbered section tags are converted to plain `SECT` tags.
5. Carries out a case conversion, so `Para` becomes `PARA`.

Although it hasn't been mentioned explicitly, XSLT defines a number of built-in (default) template rules. The complete set is listed in Section 5.8 of the spec. Mainly, they provide for the automatic copying of text and attribute nodes, and for skipping comments and processing instructions. They also dictate that inner elements are processed, even when their containing tags that don't have templates. That is the reason that the text node in the section title is processed, even though the section title is not covered by any template.

Now, run the FilterChain program, passing it the stylesheet above, the `ARTICLE` stylesheet, and the small DocBook file, in that order. The result should like this:

```
<html>
<body>
<h1 align="center">Title of my (Docbook) article</h1>
<h1>Title of Section 1.</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Conclusion

Congratulations! You have completed the XSLT tutorial! There is a lot you do with XML and XSLT, and you are now prepared to explore the many exciting possibilities that await.

Top ***Contents*** ***Index*** ***Glossary***

```
<?xml version='1.0' encoding='us-ascii'?>

<!--
    DTD for a simple "slide show".
-->

<!ELEMENT slideshow (slide+)>
<!ELEMENT slide (title, item*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT item (#PCDATA | item)* >
```

Why *WonderWidgets* are great Who *buys* WonderWidgets Market Size < predicted! Anticipated Penetration Expected Revenues Profit Margin First we fozzle the frobmorten Then we framboze the staten Finally, we frenzle the fuznaten ^ | <1> | <1> = fozzle V | <2> = framboze staten--------------------+ <3> = frenzle <2>]]>

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

<!--  A SAMPLE set of slides  -->

<!DOCTYPE slideshow SYSTEM "slideshow1a.dtd">

<slideshow
    title="Sample Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- PROCESSING INSTRUCTION -->
    <?my.presentation.Program QUERY="exec, tech, all"?>

    <!-- TITLE SLIDE -->
    <slide type="all">
       <title>Wake up to WonderWidgets!</title>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <title>Overview</title>
      <item>Why <em>WonderWidgets</em> are great</item>
      <item/>
      <item>Who <em>buys</em> WonderWidgets</item>
    </slide>

    <slide type="exec">
      <title>Financial Forecast</title>
      <item>Market Size &lt; predicted!</item>
      <item>Anticipated Penetration</item>
      <item>Expected Revenues</item>
      <item>Profit Margin </item>
    </slide>

    <slide type="tech">
      <title>How it Works</title>
      <item>First we fozzle the frobmorten</item>
      <item>Then we framboze the staten</item>
      <item>Finally, we frenzle the fuznaten</item>
      <item><![CDATA[Diagram:

           frobmorten <------------ fuznaten
              |            <3>        ^
```

```
                | <1>                       |   <1> = fozzle
                V                           |   <2> = framboze
              staten--------------------+   <3> = frenzle
                             <2>
       ]]></item>
     </slide>
 </slideshow>
```

```
<?xml version='1.0' encoding='us-ascii'?>

<!--
    DTD for a simple "slide show".
-->

<!ELEMENT slideshow (slide+)>
<!ATTLIST slideshow
            title    CDATA    #REQUIRED
            date     CDATA    #IMPLIED
            author   CDATA    "unknown"
>
<!ELEMENT slide (image?, title, item*)>
<!ATTLIST slide
            type   (tech | exec | all) #IMPLIED
>
<!ELEMENT title (#PCDATA)>
<!ELEMENT item (#PCDATA | item)* >
<!ELEMENT image EMPTY>
<!ATTLIST image
            alt    CDATA    #IMPLIED
            src    CDATA    #REQUIRED
            type   CDATA    "image/gif"
>
```

Wake up to &products;!

]> Why *&products;* are great Who *buys* &products;

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

<!--  A SAMPLE set of slides  -->

<!DOCTYPE slideshow SYSTEM "slideshow1b.dtd" [
  <!ENTITY product  "WonderWidget">
  <!ENTITY products "WonderWidgets">
]>

<!-- SUBSTITUTIONS WORK IN ATTRIBUTES, TOO -->
<slideshow
    title="&product; Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- PROCESSING INSTRUCTION -->
    <?my.presentation.Program QUERY="exec, tech, all"?>

    <!-- TITLE SLIDE -->
    <slide type="all">
       <title>Wake up to &products;!</title>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <title>Overview</title>
      <item>Why <em>&products;</em> are great</item>
      <item/>
      <item>Who <em>buys</em> &products;</item>
    </slide>

</slideshow>
```

```
Running Echo09 ../samples/slideSample06.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample06.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "WonderWidget Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <title>
            CHARS:   Wake up to
            CHARS:   WonderWidgets
            CHARS:   !
            END_ELM: </title>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
            ELEMENT: <item>
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:
            CHARS:   WonderWidgets
            END_ELM: </item>
        END_ELM: </slide>
    END_ELM: </slideshow>
END DOCUMENT
```

Wake up to &products;!

]> ©right; Why *&products;* are great Who *buys* &products;

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

<!--  A SAMPLE set of slides  -->

<!DOCTYPE slideshow SYSTEM "slideshow1b.dtd" [
  <!ENTITY product  "WonderWidget">
  <!ENTITY products "WonderWidgets">
  <!ENTITY copyright SYSTEM "copyright.xml">
]>

<!-- SUBSTITUTIONS WORK IN ATTRIBUTES, TOO -->
<slideshow
    title="&product; Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- PROCESSING INSTRUCTION -->
    <?my.presentation.Program QUERY="exec, tech, all"?>

    <!-- TITLE SLIDE -->
    <slide type="all">
       <title>Wake up to &products;!</title>
    </slide>

    <!-- COPYRIGHT SLIDE -->
    <slide type="all">
       <item>&copyright;</item>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <title>Overview</title>
      <item>Why <em>&products;</em> are great</item>
      <item/>
      <item>Who <em>buys</em> &products;</item>
    </slide>

</slideshow>
```

This is the standard copyright message that our lawyers make us put everywhere so we don't have to shell out a million bucks every time someone spills hot coffee in their lap...

```
<!--  A SAMPLE copyright  -->
This is the standard copyright message that our lawyers
make us put everywhere so we don't have to shell out a
million bucks every time someone spills hot coffee in their
lap...
```

```
Running Echo09 ../samples/slideSample07.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample07.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "WonderWidget Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <title>
            CHARS:   Wake up to
            CHARS:   WonderWidgets
            CHARS:   !
            END_ELM: </title>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <item>
            CHARS:
This is the standard copyright message that our lawyers
make us put everywhere so we don't have to shell out a
million bucks every time someone spills hot coffee in their
lap...

            END_ELM: </item>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
            ELEMENT: <item>
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:   
            CHARS:   WonderWidgets
            END_ELM: </item>
        END_ELM: </slide>
    END_ELM: </slideshow>
END DOCUMENT
```

Work in Progress!!

This tutorial is a *work in progress*. Please send us your feedback to help make it better!

> **Note:**
> To be informed of the latest releases, subscribe to the xml-announce mailing list by sending an email to listserv@java.sun.com with "`subscribe xml-interest`" and <*yourlastname*> <*yourfirstname*> in the body of the message. To unsubscribe, send email with "`unsubscribe xml-interest`" and <*yourlastname*> <*yourfirstname*> in the body of the message.

If any section of the published tutorial seems incomplete or buggy, or contains bad links, please give us feedback to help us determine what's confusing in these lessons, what seems unnecessary, and whether the lessons helped you at all. Write to us at . . .

WAIT! STOP! Before you send us an e-mail . . . you should be aware that we do not provide technical support at this address! This address is provided so that you can give us your feedback and let us know of any problems you may be having with the tutorial's *content*.

Here's where to turn for help with other problems:

- If you have a Java programming or setup question, try the The Java Developer Connection. It's the best resource we know of, and it's free! You can download early access versions of new software, scan the known bugs in the JDK, search the large database of questions and answers, and much more.
- If you had trouble downloading or unarchiving the online tutorial, or if you're having trouble browsing the tutorial on `java.sun.com`, please go to the `java.sun.com` FEEDBACK page and ask the webmaster for help.
- For more information on Java and XML in the open source community, try this link: http://xml.apache.org/

OK. Now, if you still want to send us email use this address: xml-feedback@java.sun.com. We may not be able to respond in person (we get a lot of mail!) but we thank you in advance for your help.

When sending us email, please tell us which version of the tutorial you're using. For the online tutorial, tell us the "last updated" date that's at the top of the first page. Also, please indicate which browser (include version number or date) you are using to view the tutorial, if that seems relevant.

%xhtml;

```
<?xml version='1.0' encoding='us-ascii'?>

<!--
    DTD for a simple "slide show".
-->

<!ELEMENT slideshow (slide+)>
<!ATTLIST slideshow
            title    CDATA    #REQUIRED
            date     CDATA    #IMPLIED
            author   CDATA    "unknown"
>
<!ELEMENT slide (image?, title?, item*)>
<!ATTLIST slide
            type   (tech | exec | all) #IMPLIED
>

<!-- Defines the %inline; declaration -->
<!ENTITY % xhtml SYSTEM "xhtml.dtd">
%xhtml;

<!ELEMENT title (%inline;)*>
<!ELEMENT item (%inline; | item)* >
<!ELEMENT image EMPTY>
<!ATTLIST image
            alt    CDATA    #IMPLIED
            src    CDATA    #REQUIRED
            type   CDATA    "image/gif"
>
```

Wake up to &products;!

]> ©right; Why *&products;* are great Who *buys* &products;

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

<!--  A SAMPLE set of slides  -->

<!DOCTYPE slideshow SYSTEM "slideshow2.dtd" [
  <!ENTITY product  "WonderWidget">
  <!ENTITY products "WonderWidgets">   <!-- FOR WALLY / WALLIES -->
  <!ENTITY copyright SYSTEM "copyright.xml">
]>

<!-- SUBSTITUTIONS WORK IN ATTRIBUTES, TOO -->
<slideshow
    title="&product; Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- PROCESSING INSTRUCTION -->
    <?my.presentation.Program QUERY="exec, tech, all"?>

    <!-- TITLE SLIDE -->
    <slide type="all">
       <title>Wake up to &products;!</title>
    </slide>

    <!-- TITLE SLIDE -->
    <slide type="all">
       <item>&copyright;</item>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <title>Overview</title>
      <item>Why <em>&products;</em> are great</item>
      <item/>
      <item>Who <em>buys</em> &products;</item>
    </slide>

</slideshow>
```

Why *WonderWidgets* are great Who *buys* WonderWidgets

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

<!--  A SAMPLE set of slides  -->

<slideshow
    title="Sample Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- TITLE SLIDE -->
    <slide type="all">
      <title>Wake up to WonderWidgets!</title>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <title>Overview</title>
      <item>Why <em>WonderWidgets</em> are great</item>
      <item/>
      <item>Who <em>buys</em> WonderWidgets</item>
    </slide>

</slideshow>
```

Why *WonderWidgets* are great Who *buys* WonderWidgets Market Size < predicted Anticipated Penetration Expected Revenues Profit Margin

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

<!--  A SAMPLE set of slides  -->

<slideshow
    title="Sample Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- PROCESSING INSTRUCTION -->
    <?my.presentation.Program QUERY="exec, tech, all"?>

    <!-- TITLE SLIDE -->
    <slide type="all">
       <title>Wake up to WonderWidgets!</title>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <title>Overview</title>
      <item>Why <em>WonderWidgets</em> are great</item>
      <item/>
      <item>Who <em>buys</em> WonderWidgets</item>
    </slide>

    <slide type="exec">
      <title>Financial Forecast</title>
      <item>Market Size &lt; predicted</item>
      <item>Anticipated Penetration</item>
      <item>Expected Revenues</item>
      <item>Profit Margin </item>
    </slide>

</slideshow>
```

```
Running Echo07 ../samples/slideSample03.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample03.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "Sample Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    CHARS:
    CHARS:
    PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Wake up to WonderWidgets!
            END_ELM: </title>
        CHARS:
        END_ELM: </slide>
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
        CHARS:
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:    WonderWidgets
            END_ELM: </item>
        CHARS:
        END_ELM: </slide>
    CHARS:
        ELEMENT: <slide
           ATTR: type   "exec"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Financial Forecast
            END_ELM: </title>
        CHARS:
            ELEMENT: <item>
            CHARS:   Market Size
            CHARS:   <
            CHARS:    predicted
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Anticipated Penetration
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Expected Revenues
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Profit Margin
            END_ELM: </item>
        CHARS:
        END_ELM: </slide>
    CHARS:
    END_ELM: </slideshow>
END DOCUMENT
```

Why *WonderWidgets* are great Who *buys* WonderWidgets Market Size < predicted! Anticipated Penetration Expected Revenues Profit Margin First we fozzle the frobmorten Then we framboze the staten Finally, we frenzle the fuznaten ^ | <1> | <1> = fozzle V | <2> = framboze staten--------------------+ <3> = frenzle <2>]]>

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

<!--  A SAMPLE set of slides  -->

<slideshow
    title="Sample Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- PROCESSING INSTRUCTION -->
    <?my.presentation.Program QUERY="exec, tech, all"?>

    <!-- TITLE SLIDE -->
    <slide type="all">
       <title>Wake up to WonderWidgets!</title>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <title>Overview</title>
      <item>Why <em>WonderWidgets</em> are great</item>
      <item/>
      <item>Who <em>buys</em> WonderWidgets</item>
    </slide>

    <slide type="exec">
      <title>Financial Forecast</title>
      <item>Market Size &lt; predicted!</item>
      <item>Anticipated Penetration</item>
      <item>Expected Revenues</item>
      <item>Profit Margin </item>
    </slide>

    <slide type="tech">
      <title>How it Works</title>
      <item>First we fozzle the frobmorten</item>
      <item>Then we framboze the staten</item>
      <item>Finally, we frenzle the fuznaten</item>
      <item><![CDATA[Diagram:

           frobmorten <------------ fuznaten
               |            <3>        ^
               | <1>                   |   <1> = fozzle
```

```
                V                       |   <2> = framboze
              staten--------------------+   <3> = frenzle
                             <2>
       ]]></item>
     </slide>
 </slideshow>
```

```
Running Echo07 ../samples/slideSample04.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample04.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "Sample Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    CHARS:
    CHARS:
    PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Wake up to WonderWidgets!
            END_ELM: </title>
        CHARS:
        END_ELM: </slide>
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
        CHARS:
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:    WonderWidgets
            END_ELM: </item>
        CHARS:
        END_ELM: </slide>
    CHARS:
        ELEMENT: <slide
           ATTR: type   "exec"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Financial Forecast
            END_ELM: </title>
        CHARS:
            ELEMENT: <item>
            CHARS:   Market Size
            CHARS:   <
            CHARS:    predicted!
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Anticipated Penetration
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Expected Revenues
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Profit Margin
            END_ELM: </item>
        CHARS:
        END_ELM: </slide>
    CHARS:
        ELEMENT: <slide
           ATTR: type   "tech"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   How it Works
            END_ELM: </title>
        CHARS:
            ELEMENT: <item>
            CHARS:   First we fozzle the frobmorten
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Then we framboze the staten
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Finally, we frenzle the fuznaten
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Diagram:

           frobmorten <------------ fuznaten
               |            <3>        ^
               | <1>                   |   <1> = fozzle
               V                       |   <2> = framboze
             staten--------------------+   <3> = frenzle
                            <2>

            END_ELM: </item>
        CHARS:
        END_ELM: </slide>
    CHARS:
    END_ELM: </slideshow>
END DOCUMENT
```

```java
/*
 * @(#)Echo11.java 1.5 99/02/09
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.ext.LexicalHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo11 extends DefaultHandler
   implements LexicalHandler
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        Echo11 handler = new Echo11();
        // Use the validating parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        factory.setValidating(true);
        //factory.setNamespaceAware(true);
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            XMLReader xmlReader = saxParser.getXMLReader();
            xmlReader.setProperty(
                "http://xml.org/sax/properties/lexical-handler",
                 handler
                 );
            saxParser.parse( new File(argv[0]), handler);

        } catch (SAXParseException spe) {
           // Error generated by the parser
           System.out.println("\n** Parsing error"
              + ", line " + spe.getLineNumber()
              + ", uri " + spe.getSystemId());
           System.out.println("   " + spe.getMessage() );

           // Use the contained exception, if any
           Exception  x = spe;
           if (spe.getException() != null)
               x = spe.getException();
           x.printStackTrace();

        } catch (SAXException sxe) {
           // Error generated by this application
           // (or a parser-initialization error)
           Exception  x = sxe;
           if (sxe.getException() != null)
               x = sxe.getException();
           x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }

        System.exit(0);
    }

    static private Writer  out;
    private String indentString = "    "; // Amount to indent
    private int indentLevel = 0;

    //===========================================================
    // SAX DocumentHandler methods
    //===========================================================

    public void setDocumentLocator(Locator l)
    {
        // Save this to resolve relative URIs or to give diagnostics.
        try {
          out.write("LOCATOR");
          out.write("\n SYS ID: " + l.getSystemId() );
          out.flush();
        } catch (IOException e) {
            // Ignore errors
        }
    }

    public void startDocument()
    throws SAXException
    {
        nl();
        nl();
        emit("START DOCUMENT");
        nl();
        emit("<?xml version='1.0' encoding='UTF-8'?>");
    }

    public void endDocument()
    throws SAXException
    {
        nl(); emit("END DOCUMENT");
        try {
            nl();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        indentLevel++;
        nl(); emit("ELEMENT: ");
        String eName = lName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                nl();
                emit("   ATTR: ");
                emit(aName);
                emit("\t\"");
                emit(attrs.getValue(i));
                emit("\"");
            }
        }
        if (attrs.getLength() > 0) nl();
        emit(">");
    }

    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          )
    throws SAXException
    {
        nl();
        emit("END_ELM: ");
        emit("</"+sName+">");
        indentLevel--;
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        nl(); emit("CHARS:   ");
        String s = new String(buf, offset, len);
        if (!s.trim().equals("")) emit(s);
    }

    public void ignorableWhitespace(char buf[], int offset, int len)
    throws SAXException
    {
        // Ignore it
    }

    public void processingInstruction(String target, String data)
    throws SAXException
    {
        nl();
        emit("PROCESS: ");
        emit("<?"+target+" "+data+"?>");
    }

    //===========================================================
    // SAX ErrorHandler methods
    //===========================================================

    // treat validation errors as fatal
    public void error(SAXParseException e)
    throws SAXParseException
    {
        throw e;
    }

    // dump warnings too
    public void warning(SAXParseException err)
    throws SAXParseException
    {
        System.out.println("** Warning"
            + ", line " + err.getLineNumber()
            + ", uri " + err.getSystemId());
        System.out.println("   " + err.getMessage());
    }

    //===========================================================
    // LexicalEventListener methods
    //===========================================================

    public void comment(char[] ch, int start, int length)
    throws SAXException
    {
       String text = new String(ch, start, length);
       nl(); emit("COMMENT: "+text);
    }

    public void startCDATA()
    throws SAXException
    {
    }

    public void endCDATA()
    throws SAXException
    {
    }
    public void startEntity(java.lang.String name)
    throws SAXException
    {
    }

    public void endEntity(java.lang.String name)
    throws SAXException
    {
    }

    public void startDTD(String name, String publicId, String systemId)
    throws SAXException
    {
    }

    public void endDTD()
    throws SAXException
    {
    }

    //===========================================================
    // Utility Methods ...
    //===========================================================

    // Wrap I/O exceptions in SAX exceptions, to
    // suit handler signature requirements
    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    // Start a new line
    // and indent the next line appropriately
    private void nl()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
            for (int i=0; i < indentLevel; i++) out.write(indentString);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
}
```

```
Running Echo11 ../samples/slideSample09.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample09.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
COMMENT:   A SAMPLE set of slides
COMMENT:  FOR WALLY / WALLIES
COMMENT:
    DTD for a simple "slide show".

COMMENT:  Defines the %inline; declaration
COMMENT:
    This DTD does some of what the W3C is getting ready to do with its
    "XHTML" work (nee "Voyager").  It differs from the current WG draft
    because it uses namespaces correctly (!), and it isn't as complete
    even for HTML 3.2 support (much less 4.0) or, probably, correct.

    Note that what XHTML needs to do is become modular enough that XHTML
    can be used as a mixin with other document types, including either
    "the whole megillah" or just selected modules (e.g. omitting tables).
    That must work both ways ... other things as mixins to XHTML, and
    XHTML as a mixin to other things.

    THIS WILL BE REPLACED WITH A BETTER DTD AT SOME POINT.

COMMENT:  SUBSTITUTIONS WORK IN ATTRIBUTES, TOO
    ELEMENT: <slideshow
       ATTR: title      "WonderWidget Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    COMMENT:  PROCESSING INSTRUCTION
    PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
    COMMENT:  TITLE SLIDE
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <slide-title>
            CHARS:   Wake up to
            CHARS:   WonderWidgets
            CHARS:   !
            END_ELM: </slide-title>
        END_ELM: </slide>
    COMMENT:  TITLE SLIDE
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <item>
            COMMENT:   A SAMPLE copyright
            CHARS:
This is the standard copyright message that our lawyers
```

```
make us put everywhere so we don't have to shell out a
million bucks every time someone spills hot coffee in their
lap...

            END_ELM: </item>
        END_ELM: </slide>
    COMMENT:  OVERVIEW
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <slide-title>
            CHARS:   Overview
            END_ELM: </slide-title>
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
            ELEMENT: <item>
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:
            CHARS:   WonderWidgets
            END_ELM: </item>
        END_ELM: </slide>
    END_ELM: </slideshow>
END DOCUMENT
```

```
/*
 * @(#)Echo12.java 1.5 99/02/09
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.ext.LexicalHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo12 extends DefaultHandler
   implements LexicalHandler
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        Echo12 handler = new Echo12();
        // Use the validating parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        factory.setValidating(true);
        //factory.setNamespaceAware(true);
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
```

```
          XMLReader xmlReader = saxParser.getXMLReader();
          xmlReader.setProperty(
              "http://xml.org/sax/properties/lexical-handler",
               handler
               );
          saxParser.parse( new File(argv[0]), handler);

      } catch (SAXParseException spe) {
         // Error generated by the parser
         System.out.println("\n** Parsing error"
            + ", line " + spe.getLineNumber()
            + ", uri " + spe.getSystemId());
         System.out.println("   " + spe.getMessage() );

         // Use the contained exception, if any
         Exception  x = spe;
         if (spe.getException() != null)
             x = spe.getException();
         x.printStackTrace();

      } catch (SAXException sxe) {
         // Error generated by this application
         // (or a parser-initialization error)
         Exception  x = sxe;
         if (sxe.getException() != null)
             x = sxe.getException();
         x.printStackTrace();

      } catch (ParserConfigurationException pce) {
          // Parser with specified options can't be built
          pce.printStackTrace();

      } catch (IOException ioe) {
         // I/O error
         ioe.printStackTrace();
      }

      System.exit(0);
  }

  static private Writer  out;
  private String indentString = "    "; // Amount to indent
  private int indentLevel = 0;

  //===========================================================
  // SAX DocumentHandler methods
  //===========================================================

  public void setDocumentLocator(Locator l)
  {
      // Save this to resolve relative URIs or to give diagnostics.
      try {
        out.write("LOCATOR");
        out.write("\n SYS ID: " + l.getSystemId() );
        out.flush();
      } catch (IOException e) {
          // Ignore errors
      }
  }

  public void startDocument()
  throws SAXException
```

```
    {
        nl();
        nl();
        emit("START DOCUMENT");
        nl();
        emit("<?xml version='1.0' encoding='UTF-8'?>");
    }

    public void endDocument()
    throws SAXException
    {
        nl(); emit("END DOCUMENT");
        try {
            nl();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        indentLevel++;
        nl(); emit("ELEMENT: ");
        String eName = lName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                nl();
                emit("   ATTR: ");
                emit(aName);
                emit("\t\"");
                emit(attrs.getValue(i));
                emit("\"");
            }
        }
        if (attrs.getLength() > 0) nl();
        emit(">");
    }

    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          )
    throws SAXException
    {
        nl();
        emit("END_ELM: ");
        emit("</"+sName+">");
        indentLevel--;
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        nl(); emit("CHARS:   ");
        String s = new String(buf, offset, len);
        if (!s.trim().equals("")) emit(s);
    }

    public void ignorableWhitespace(char buf[], int offset, int len)
    throws SAXException
    {
        // Ignore it
    }

    public void processingInstruction(String target, String data)
    throws SAXException
    {
        nl();
        emit("PROCESS: ");
        emit("<?"+target+" "+data+"?>");
    }

    //===========================================================
    // SAX ErrorHandler methods
    //===========================================================

    // treat validation errors as fatal
    public void error(SAXParseException e)
    throws SAXParseException
    {
        throw e;
    }

    // dump warnings too
    public void warning(SAXParseException err)
    throws SAXParseException
    {
        System.out.println("** Warning"
            + ", line " + err.getLineNumber()
            + ", uri " + err.getSystemId());
        System.out.println("   " + err.getMessage());
    }

    //===========================================================
    // LexicalEventListener methods
    //===========================================================

    public void comment(char[] ch, int start, int length)
    throws SAXException
    {
    }

    public void startCDATA()
    throws SAXException
    {
       nl(); emit("START CDATA SECTION");
    }

    public void endCDATA()
    throws SAXException
    {
       nl(); emit("END CDATA SECTION");
    }
    public void startEntity(java.lang.String name)
    throws SAXException
    {
       nl(); emit("START ENTITY: "+name);
    }

    public void endEntity(java.lang.String name)
    throws SAXException
    {
       nl(); emit("END ENTITY: "+name);
    }

    public void startDTD(String name, String publicId, String systemId)
    throws SAXException
    {
       nl(); emit("START DTD: "+name
               +"\n           publicId=" + publicId
               +"\n           systemId=" + systemId);
    }

    public void endDTD()
    throws SAXException
    {
       nl(); emit("END DTD");
    }

    //===========================================================
    // Utility Methods ...
    //===========================================================

    // Wrap I/O exceptions in SAX exceptions, to
    // suit handler signature requirements
    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    // Start a new line
    // and indent the next line appropriately
    private void nl()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
            for (int i=0; i < indentLevel; i++) out.write(indentString);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
}
```

]> Wake up to &products;! ©right; Overview Why *&products;* are great Who *buys* &products; How it Works First we fozzle the frobmorten Then we framboze the staten Finally, we frenzle the fuznaten ^ | <1> | <1> = fozzle V | <2> = framboze staten--------------------+ <3> = frenzle <2>]]>

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

<!--  A SAMPLE set of slides  -->

<!DOCTYPE slideshow SYSTEM "slideshow3.dtd" [
  <!ENTITY product  "WonderWidget">
  <!ENTITY products "WonderWidgets">   <!-- FOR WALLY / WALLIES -->
  <!ENTITY copyright SYSTEM "copyright.xml">
]>

<!-- SUBSTITUTIONS WORK IN ATTRIBUTES, TOO -->
<slideshow
    title="&product; Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- PROCESSING INSTRUCTION -->
    <?my.presentation.Program QUERY="exec, tech, all"?>

    <!-- TITLE SLIDE -->
    <slide type="all">
       <slide-title>Wake up to &products;!</slide-title>
    </slide>

    <!-- TITLE SLIDE -->
    <slide type="all">
       <item>&copyright;</item>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <slide-title>Overview</slide-title>
      <item>Why <em>&products;</em> are great</item>
      <item/>
      <item>Who <em>buys</em> &products;</item>
    </slide>

    <slide type="tech">
      <slide-title>How it Works</slide-title>
      <item>First we fozzle the frobmorten</item>
      <item>Then we framboze the staten</item>
      <item>Finally, we frenzle the fuznaten</item>
      <item><![CDATA[Diagram:
```

```
    frobmorten <------------ fuznaten
        |            <3>        ^
        | <1>                   |   <1> = fozzle
        V                       |   <2> = framboze
      staten--------------------+   <3> = frenzle
                     <2>
      ]]></item>
    </slide>

</slideshow>
```

```
Running Echo12 ../samples/slideSample10.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample10.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
START DTD: slideshow
           publicId=null

systemId=file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideshow3.dtd
END DTD
    ELEMENT: <slideshow
       ATTR: title      "WonderWidget Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <slide-title>
            CHARS:   Wake up to
            START ENTITY: products
            CHARS:   WonderWidgets
            END ENTITY: products
            CHARS:   !
            END_ELM: </slide-title>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <item>
            START ENTITY: copyright
            CHARS:
This is the standard copyright message that our lawyers
make us put everywhere so we don't have to shell out a
million bucks every time someone spills hot coffee in their
lap...

            END ENTITY: copyright
            END_ELM: </item>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <slide-title>
            CHARS:   Overview
            END_ELM: </slide-title>
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                START ENTITY: products
                CHARS:   WonderWidgets
                END ENTITY: products
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
            ELEMENT: <item>
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:
            START ENTITY: products
            CHARS:   WonderWidgets
            END ENTITY: products
            END_ELM: </item>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "tech"
        >
            ELEMENT: <slide-title>
            CHARS:   How it Works
            END_ELM: </slide-title>
            ELEMENT: <item>
            CHARS:   First we fozzle the frobmorten
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Then we framboze the staten
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Finally, we frenzle the fuznaten
            END_ELM: </item>
            ELEMENT: <item>
            START CDATA SECTION
            CHARS:   Diagram:

    frobmorten <------------ fuznaten
        |            <3>        ^
        | <1>                   |   <1> = fozzle
        V                       |   <2> = framboze
      staten--------------------+   <3> = frenzle
                     <2>

            END CDATA SECTION
            END_ELM: </item>
        END_ELM: </slide>
    END_ELM: </slideshow>
END DOCUMENT
```

```
Running Echo07 ../samples/slideSample05.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample05.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "Sample Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <title>
            CHARS:   Wake up to WonderWidgets!
            END_ELM: </title>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
            ELEMENT: <item>
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:    WonderWidgets
            END_ELM: </item>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "exec"
        >
            ELEMENT: <title>
            CHARS:   Financial Forecast
            END_ELM: </title>
            ELEMENT: <item>
            CHARS:   Market Size
            CHARS:   <
            CHARS:    predicted!
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Anticipated Penetration
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Expected Revenues
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Profit Margin
            END_ELM: </item>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "tech"
        >
            ELEMENT: <title>
            CHARS:   How it Works
            END_ELM: </title>
            ELEMENT: <item>
            CHARS:   First we fozzle the frobmorten
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Then we framboze the staten
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Finally, we frenzle the fuznaten
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Diagram:

           frobmorten <------------ fuznaten
               |            <3>        ^
               | <1>                   |   <1> = fozzle
               V                       |   <2> = framboze
             staten--------------------+   <3> = frenzle
                            <2>

            END_ELM: </item>
        END_ELM: </slide>
    END_ELM: </slideshow>
END DOCUMENT
```

```java
/*
 * @(#)Echo08.java 1.5 99/02/09
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo08 extends DefaultHandler
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        DefaultHandler handler = new Echo08();
        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler);

        } catch (SAXParseException spe) {
           // Error generated by the parser
           System.out.println("\n** Parsing error"
              + ", line " + spe.getLineNumber()
              + ", uri " + spe.getSystemId());
           System.out.println("   " + spe.getMessage() );

           // Use the contained exception, if any
           Exception  x = spe;
           if (spe.getException() != null)
               x = spe.getException();
           x.printStackTrace();

        } catch (SAXException sxe) {
           // Error generated by this application
           // (or a parser-initialization error)
           Exception  x = sxe;
           if (sxe.getException() != null)
               x = sxe.getException();
           x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }

        System.exit(0);
    }

    static private Writer  out;
    private String indentString = "    "; // Amount to indent
    private int indentLevel = 0;

    //===========================================================
    // SAX DocumentHandler methods
    //===========================================================

    public void setDocumentLocator(Locator l)
    {
        // Save this to resolve relative URIs or to give diagnostics.
        try {
          out.write("LOCATOR");
          out.write("\n SYS ID: " + l.getSystemId() );
          out.flush();
        } catch (IOException e) {
            // Ignore errors
        }
    }

    public void startDocument()
    throws SAXException
    {
        nl();
        nl();
        emit("START DOCUMENT");
        nl();
        emit("<?xml version='1.0' encoding='UTF-8'?>");
    }

    public void endDocument()
    throws SAXException
    {
        nl(); emit("END DOCUMENT");
        try {
            nl();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        indentLevel++;
        nl(); emit("ELEMENT: ");
        String eName = lName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                nl();
                emit("   ATTR: ");
                emit(aName);
                emit("\t\"");
                emit(attrs.getValue(i));
                emit("\"");
            }
        }
        if (attrs.getLength() > 0) nl();
        emit(">");
    }

    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          )
    throws SAXException
    {
        nl();
        emit("END_ELM: ");
        emit("</"+sName+">");
        indentLevel--;
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        nl(); emit("CHARS:   ");
        String s = new String(buf, offset, len);
        if (!s.trim().equals("")) emit(s);
    }

    public void ignorableWhitespace(char buf[], int offset, int len)
    throws SAXException
    {
        nl(); emit("IGNORABLE");
    }

    public void processingInstruction(String target, String data)
    throws SAXException
    {
        nl();
        emit("PROCESS: ");
        emit("<?"+target+" "+data+"?>");
    }

    //===========================================================
    // SAX ErrorHandler methods
    //===========================================================

    // treat validation errors as fatal
    public void error(SAXParseException e)
    throws SAXParseException
    {
        throw e;
    }

    // dump warnings too
    public void warning(SAXParseException err)
    throws SAXParseException
    {
        System.out.println("** Warning"
            + ", line " + err.getLineNumber()
            + ", uri " + err.getSystemId());
        System.out.println("   " + err.getMessage());
    }

    //===========================================================
    // Utility Methods ...
    //===========================================================

    // Wrap I/O exceptions in SAX exceptions, to
    // suit handler signature requirements
    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    // Start a new line
    // and indent the next line appropriately
    private void nl()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
            for (int i=0; i < indentLevel; i++) out.write(indentString);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
```

```
}
```

```
Running Echo08 ../samples/slideSample05.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample05.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "Sample Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    IGNORABLE
    IGNORABLE
    PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
    IGNORABLE
    IGNORABLE
        ELEMENT: <slide
           ATTR: type   "all"
        >
        IGNORABLE
            ELEMENT: <title>
            CHARS:   Wake up to WonderWidgets!
            END_ELM: </title>
        IGNORABLE
        END_ELM: </slide>
    IGNORABLE
    IGNORABLE
        ELEMENT: <slide
           ATTR: type   "all"
        >
        IGNORABLE
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
        IGNORABLE
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
        IGNORABLE
            ELEMENT: <item>
            END_ELM: </item>
        IGNORABLE
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:    WonderWidgets
            END_ELM: </item>
        IGNORABLE
        END_ELM: </slide>
    IGNORABLE
        ELEMENT: <slide
           ATTR: type   "exec"
        >
        IGNORABLE
            ELEMENT: <title>
            CHARS:   Financial Forecast
            END_ELM: </title>
        IGNORABLE
            ELEMENT: <item>
            CHARS:   Market Size
            CHARS:   <
            CHARS:    predicted!
            END_ELM: </item>
        IGNORABLE
            ELEMENT: <item>
            CHARS:   Anticipated Penetration
            END_ELM: </item>
        IGNORABLE
            ELEMENT: <item>
            CHARS:   Expected Revenues
            END_ELM: </item>
        IGNORABLE
            ELEMENT: <item>
            CHARS:   Profit Margin
            END_ELM: </item>
        IGNORABLE
        END_ELM: </slide>
    IGNORABLE
        ELEMENT: <slide
           ATTR: type   "tech"
        >
        IGNORABLE
            ELEMENT: <title>
            CHARS:   How it Works
            END_ELM: </title>
        IGNORABLE
            ELEMENT: <item>
            CHARS:   First we fozzle the frobmorten
            END_ELM: </item>
        IGNORABLE
            ELEMENT: <item>
            CHARS:   Then we framboze the staten
            END_ELM: </item>
        IGNORABLE
            ELEMENT: <item>
            CHARS:   Finally, we frenzle the fuznaten
            END_ELM: </item>
        IGNORABLE
            ELEMENT: <item>
            CHARS:   Diagram:

           frobmorten <------------ fuznaten
               |            <3>        ^
               | <1>                   |   <1> = fozzle
               V                       |   <2> = framboze
             staten--------------------+   <3> = frenzle
                            <2>

            END_ELM: </item>
        IGNORABLE
        END_ELM: </slide>
    IGNORABLE
    END_ELM: </slideshow>
END DOCUMENT
```

```
/*
 * @(#)Echo09.java 1.5 99/02/09
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo09 extends DefaultHandler
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        DefaultHandler handler = new Echo09();
        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler);

        } catch (SAXParseException spe) {
           // Error generated by the parser
```

```
            System.out.println("\n** Parsing error"
               + ", line " + spe.getLineNumber()
               + ", uri " + spe.getSystemId());
            System.out.println("   " + spe.getMessage() );

            // Use the contained exception, if any
            Exception  x = spe;
            if (spe.getException() != null)
                x = spe.getException();
            x.printStackTrace();

        } catch (SAXException sxe) {
            // Error generated by this application
            // (or a parser-initialization error)
            Exception  x = sxe;
            if (sxe.getException() != null)
                x = sxe.getException();
            x.printStackTrace();

        } catch (ParserConfigurationException pce) {
             // Parser with specified options can't be built
             pce.printStackTrace();

        } catch (IOException ioe) {
            // I/O error
            ioe.printStackTrace();
        }

        System.exit(0);
    }

    static private Writer  out;
    private String indentString = "    "; // Amount to indent
    private int indentLevel = 0;

    //===========================================================
    // SAX DocumentHandler methods
    //===========================================================

    public void setDocumentLocator(Locator l)
    {
        // Save this to resolve relative URIs or to give diagnostics.
        try {
          out.write("LOCATOR");
          out.write("\n SYS ID: " + l.getSystemId() );
          out.flush();
        } catch (IOException e) {
            // Ignore errors
        }
    }

    public void startDocument()
    throws SAXException
    {
        nl();
        nl();
        emit("START DOCUMENT");
        nl();
        emit("<?xml version='1.0' encoding='UTF-8'?>");
    }

    public void endDocument()
    throws SAXException
    {
        nl(); emit("END DOCUMENT");
        try {
            nl();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        indentLevel++;
        nl(); emit("ELEMENT: ");
        String eName = lName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                nl();
                emit("   ATTR: ");
                emit(aName);
                emit("\t\"");
                emit(attrs.getValue(i));
                emit("\"");
            }
        }
        if (attrs.getLength() > 0) nl();
        emit(">");
    }

    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          )
    throws SAXException
    {
        nl();
        emit("END_ELM: ");
        emit("</"+sName+">");
        indentLevel--;
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        nl(); emit("CHARS:   ");
        String s = new String(buf, offset, len);
        if (!s.trim().equals("")) emit(s);
    }

    public void ignorableWhitespace(char buf[], int offset, int len)
    throws SAXException
    {
        // Ignore it
    }

    public void processingInstruction(String target, String data)
    throws SAXException
    {
        nl();
        emit("PROCESS: ");
        emit("<?"+target+" "+data+"?>");
    }

    //===========================================================
    // SAX ErrorHandler methods
    //===========================================================

    // treat validation errors as fatal
    public void error(SAXParseException e)
    throws SAXParseException
    {
        throw e;
    }

    // dump warnings too
    public void warning(SAXParseException err)
    throws SAXParseException
    {
        System.out.println("** Warning"
            + ", line " + err.getLineNumber()
            + ", uri " + err.getSystemId());
        System.out.println("   " + err.getMessage());
    }

    //===========================================================
    // Utility Methods ...
    //===========================================================

    // Wrap I/O exceptions in SAX exceptions, to
    // suit handler signature requirements
    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    // Start a new line
    // and indent the next line appropriately
    private void nl()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
            for (int i=0; i < indentLevel; i++) out.write(indentString);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
```

```
}
```

```java
/*
 * @(#)Echo10.java 1.5 99/02/09
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo10 extends DefaultHandler
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        DefaultHandler handler = new Echo10();
        // Use the validating parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        factory.setValidating(true);
        //factory.setNamespaceAware(true);
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler);
        } catch (SAXParseException spe) {
           // Error generated by the parser
           System.out.println("\n** Parsing error"
              + ", line " + spe.getLineNumber()
              + ", uri " + spe.getSystemId());
           System.out.println("   " + spe.getMessage() );

           // Use the contained exception, if any
           Exception  x = spe;
           if (spe.getException() != null)
               x = spe.getException();
           x.printStackTrace();

        } catch (SAXException sxe) {
           // Error generated by this application
           // (or a parser-initialization error)
           Exception  x = sxe;
           if (sxe.getException() != null)
               x = sxe.getException();
           x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }

        System.exit(0);
    }

    static private Writer  out;
    private String indentString = "    "; // Amount to indent
    private int indentLevel = 0;

    //===========================================================
    // SAX DocumentHandler methods
    //===========================================================

    public void setDocumentLocator(Locator l)
    {
        // Save this to resolve relative URIs or to give diagnostics.
        try {
          out.write("LOCATOR");
          out.write("\n SYS ID: " + l.getSystemId() );
          out.flush();
        } catch (IOException e) {
            // Ignore errors
        }
    }

    public void startDocument()
    throws SAXException
    {
        nl();
        nl();
        emit("START DOCUMENT");
        nl();
        emit("<?xml version='1.0' encoding='UTF-8'?>");
    }
    public void endDocument()
    throws SAXException
    {
        nl(); emit("END DOCUMENT");
        try {
            nl();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        indentLevel++;
        nl(); emit("ELEMENT: ");
        String eName = lName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                nl();
                emit("   ATTR: ");
                emit(aName);
                emit("\t\"");
                emit(attrs.getValue(i));
                emit("\"");
            }
        }
        if (attrs.getLength() > 0) nl();
        emit(">");
    }

    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          )
    throws SAXException
    {
        nl();
        emit("END_ELM: ");
        emit("</"+sName+">");
        indentLevel--;
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        nl(); emit("CHARS:   ");
        String s = new String(buf, offset, len);
        if (!s.trim().equals("")) emit(s);
    }

    public void ignorableWhitespace(char buf[], int offset, int len)
    throws SAXException
    {
        // Ignore it
    }

    public void processingInstruction(String target, String data)
    throws SAXException
    {
        nl();
        emit("PROCESS: ");
        emit("<?"+target+" "+data+"?>");
    }

    //===========================================================
    // SAX ErrorHandler methods
    //===========================================================

    // treat validation errors as fatal
    public void error(SAXParseException e)
    throws SAXParseException
    {
        throw e;
    }

    // dump warnings too
    public void warning(SAXParseException err)
    throws SAXParseException
    {
        System.out.println("** Warning"
            + ", line " + err.getLineNumber()
            + ", uri " + err.getSystemId());
        System.out.println("   " + err.getMessage());
    }

    //===========================================================
    // Utility Methods ...
    //===========================================================

    // Wrap I/O exceptions in SAX exceptions, to
    // suit handler signature requirements
    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    // Start a new line
    // and indent the next line appropriately
    private void nl()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
            for (int i=0; i < indentLevel; i++) out.write(indentString);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
}
```

```
Running Echo10 ../samples/slideSample01.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample01.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>** Warning, line 5, uri
file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample01.xml
   Valid documents must have a <!DOCTYPE declaration.

** Parsing error, line 5, uri
file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample01.xml
   Element type "slideshow" is not declared.
org.xml.sax.SAXParseException: Element type "slideshow" is not declared.
        at org.apache.crimson.parser.Parser2.error(Parser2.java:3013)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1308)
        at org.apache.crimson.parser.Parser2.parseInternal(Parser2.java:499)
        at org.apache.crimson.parser.Parser2.parse(Parser2.java:304)
        at org.apache.crimson.parser.XMLReaderImpl.parse(XMLReaderImpl.java:433)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:346)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:286)
        at Echo10.main(Echo10.java:62)
```

```
Running Echo10 ../samples/slideSample06.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample06.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "WonderWidget Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <title>
            CHARS:   Wake up to
            CHARS:   WonderWidgets
            CHARS:   !
            END_ELM: </title>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
            ELEMENT: <item>
            CHARS:   Why
** Parsing error, line 28, uri
file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample06.xml
   Element "item" does not allow "em" -- (#PCDATA|item)
org.xml.sax.SAXParseException: Element "item" does not allow "em" -- (#PCDATA|item)
        at org.apache.crimson.parser.Parser2.error(Parser2.java:3013)
        at
org.apache.crimson.parser.ValidatingParser$MixedValidator.consume(ValidatingParser.java:327)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1303)
        at org.apache.crimson.parser.Parser2.content(Parser2.java:1700)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1468)
        at org.apache.crimson.parser.Parser2.content(Parser2.java:1700)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1468)
        at org.apache.crimson.parser.Parser2.content(Parser2.java:1700)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1468)
        at org.apache.crimson.parser.Parser2.parseInternal(Parser2.java:499)
        at org.apache.crimson.parser.Parser2.parse(Parser2.java:304)
        at org.apache.crimson.parser.XMLReaderImpl.parse(XMLReaderImpl.java:433)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:346)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:286)
        at Echo10.main(Echo10.java:62)
```

```
Running Echo10 ../samples/slideSample07.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample07.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "WonderWidget Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <title>
            CHARS:   Wake up to
            CHARS:   WonderWidgets
            CHARS:   !
            END_ELM: </title>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "all"
        >
** Parsing error, line 28, uri
file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample07.xml
   Element "slide" does not allow "item" here.
org.xml.sax.SAXParseException: Element "slide" does not allow "item" here.
        at org.apache.crimson.parser.Parser2.error(Parser2.java:3013)
        at
org.apache.crimson.parser.ValidatingParser$ChildrenValidator.consume(ValidatingParser.java:349)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1303)
        at org.apache.crimson.parser.Parser2.content(Parser2.java:1700)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1468)
        at org.apache.crimson.parser.Parser2.content(Parser2.java:1700)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1468)
        at org.apache.crimson.parser.Parser2.parseInternal(Parser2.java:499)
        at org.apache.crimson.parser.Parser2.parse(Parser2.java:304)
        at org.apache.crimson.parser.XMLReaderImpl.parse(XMLReaderImpl.java:433)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:346)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:286)
        at Echo10.main(Echo10.java:62)
```

```
#!/bin/csh
echo Compiling $*

if ("$?JAXP" == "0") set JAXP=""       # If not defined, define it.
if ("$JAXP" == "") then
  # JAXP was not defined or has no value
  echo Using JAXP bundles installed as standard extensions.
  javac $*
else
  echo Using JAXP variable to access bundles at $JAXP
  set CP = .:${JAXP}/jaxp.jar:${JAXP}/crimson.jar:${JAXP}/xalan.jar
  javac -classpath $CP $*
endif
```

```
#!/bin/csh

echo Running $*
if (! $?JAXP) then
  #  Using JAXP bundles installed as standard extensions.
  set CP = "."
else
  # Using JAXP variable to access bundles at $JAXP
  set CP = ".:${JAXP}/jaxp.jar:${JAXP}/crimson.jar:${JAXP}/xalan.jar"
endif
java -classpath $CP $*
```

```
echo Compiling %*%

if "%JAXP%" == "" goto DEFAULT

echo Using JAXP variable to access bundles at %JAXP%
set CP=.;%JAXP%\jaxp.jar;%JAXP%\crimson.jar;%JAXP%\xalan.jar
javac -classpath %CP%  %*%
goto EXIT

:DEFAULT
echo Using JAXP bundles installed as standard extensions.
javac %*%

:EXIT
```

```
echo Running %1%.java

if "%JAXP%" == "" goto DEFAULT

echo Using JAXP variable to access bundles at %JAXP%
set CP=.;%JAXP%\jaxp.jar;%JAXP%\crimson.jar;%JAXP%\xalan.jar
goto RUN

:DEFAULT
echo Using JAXP bundles installed as standard extensions.
set CP="."

:RUN
java -classpath %CP%  %*%
```

```java
/*
 * @(#)Echo01.java 1.5 99/02/09
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo01 extends DefaultHandler
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        DefaultHandler handler = new Echo01();
        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler);

        } catch (Throwable t) {
            t.printStackTrace();
        }
        System.exit(0);
    }

    static private Writer  out;

    //===========================================================
    // SAX DocumentHandler methods
    //===========================================================

    public void startDocument()
    throws SAXException
    {
        emit("<?xml version='1.0' encoding='UTF-8'?>");
        nl();
    }

    public void endDocument()
    throws SAXException
    {
        try {
            nl();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        String eName = lName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                emit(" ");
                emit(aName+"=\""+attrs.getValue(i)+"\"");
            }
        }
        emit(">");
    }

    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          )
    throws SAXException
    {
        emit("</"+sName+">");
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        String s = new String(buf, offset, len);
        emit(s);
    }

    //===========================================================
    // Utility Methods ...
    //===========================================================

    // Wrap I/O exceptions in SAX exceptions, to
    // suit handler signature requirements
    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    // Start a new line
    private void nl()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
}
```

```
Running Echo01 ../samples/slideSample01.xml
<?xml version='1.0' encoding='UTF-8'?>
<slideshow title="Sample Slide Show" date="Date of publication" author="Yours Truly">


    <slide type="all">
      <title>Wake up to WonderWidgets!</title>
    </slide>


    <slide type="all">
      <title>Overview</title>
      <item>Why <em>WonderWidgets</em> are great</item>
      <item></item>
      <item>Who <em>buys</em> WonderWidgets</item>
    </slide>

</slideshow>
```

```java
/*
 * @(#)Echo02.java 1.5 99/02/09
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo02 extends DefaultHandler
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        DefaultHandler handler = new Echo02();
        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler);

        } catch (Throwable t) {
            t.printStackTrace();
        }
        System.exit(0);
    }

    static private Writer  out;

    //===========================================================
    // SAX DocumentHandler methods
    //===========================================================

    public void startDocument()
    throws SAXException
    {
        nl();
        nl();
        emit("START DOCUMENT");
        nl();
        emit("<?xml version='1.0' encoding='UTF-8'?>");
    }

    public void endDocument()
    throws SAXException
    {
        nl(); emit("END DOCUMENT");
        try {
            nl();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        nl(); emit("ELEMENT: ");
        String eName = lName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                nl();
                emit("   ATTR: ");
                emit(aName);
                emit("\t\"");
                emit(attrs.getValue(i));
                emit("\"");
            }
        }
        if (attrs.getLength() > 0) nl();
        emit(">");
    }

    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          )
    throws SAXException
    {
        nl();
        emit("END_ELM: ");
        emit("</"+sName+">");
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        nl(); emit("CHARS: |");
        String s = new String(buf, offset, len);
        emit(s);
        emit("|");
    }

    //===========================================================
    // Utility Methods ...
    //===========================================================

    // Wrap I/O exceptions in SAX exceptions, to
    // suit handler signature requirements
    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    // Start a new line
    private void nl()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
}
```

```
Running Echo02 ../samples/slideSample01.xml


START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
ELEMENT: <slideshow
   ATTR: title  "Sample Slide Show"
   ATTR: date   "Date of publication"
   ATTR: author "Yours Truly"
>
CHARS: |

    |
CHARS: |
    |
ELEMENT: <slide
   ATTR: type   "all"
>
CHARS: |
      |
ELEMENT: <title>
CHARS: |Wake up to WonderWidgets!|
END_ELM: </title>
CHARS: |
    |
END_ELM: </slide>
CHARS: |

    |
CHARS: |
    |
ELEMENT: <slide
   ATTR: type   "all"
>
CHARS: |
      |
ELEMENT: <title>
CHARS: |Overview|
END_ELM: </title>
CHARS: |
      |
ELEMENT: <item>
CHARS: |Why |
ELEMENT: <em>
```

```
CHARS: |WonderWidgets|
END_ELM: </em>
CHARS: | are great|
END_ELM: </item>
CHARS: |
      |
ELEMENT: <item>
END_ELM: </item>
CHARS: |
      |
ELEMENT: <item>
CHARS: |Who |
ELEMENT: <em>
CHARS: |buys|
END_ELM: </em>
CHARS: | WonderWidgets|
END_ELM: </item>
CHARS: |
    |
END_ELM: </slide>
CHARS: |

|
END_ELM: </slideshow>
END DOCUMENT
```

```
/*
 * @(#)Echo03.java 1.5 99/02/09
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo03 extends DefaultHandler
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        DefaultHandler handler = new Echo03();
        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler);

        } catch (Throwable t) {
            t.printStackTrace();
        }
        System.exit(0);
    }

    static private Writer  out;
    private String indentString = "    "; // Amount to indent
    private int indentLevel = 0;

    //===========================================================
    // SAX DocumentHandler methods
    //===========================================================

    public void startDocument()
    throws SAXException
    {
        nl();
        nl();
        emit("START DOCUMENT");
        nl();
        emit("<?xml version='1.0' encoding='UTF-8'?>");
    }

    public void endDocument()
    throws SAXException
    {
        nl(); emit("END DOCUMENT");
        try {
            nl();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        indentLevel++;
        nl(); emit("ELEMENT: ");
        String eName = lName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                nl();
                emit("   ATTR: ");
                emit(aName);
                emit("\t\"");
                emit(attrs.getValue(i));
                emit("\"");
            }
        }
        if (attrs.getLength() > 0) nl();
        emit(">");
    }

    public void endElement(String namespaceURI,
                             String sName, // simple name
                             String qName  // qualified name
                            )
    throws SAXException
    {
        nl();
        emit("END_ELM: ");
        emit("</"+sName+">");
        indentLevel--;
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        nl(); emit("CHARS:   ");
        String s = new String(buf, offset, len);
        if (!s.trim().equals("")) emit(s);
    }

    //===========================================================
    // Utility Methods ...
    //===========================================================

    // Wrap I/O exceptions in SAX exceptions, to
    // suit handler signature requirements
    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    // Start a new line
    // and indent the next line appropriately
    private void nl()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
            for (int i=0; i < indentLevel; i++) out.write(indentString);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
}
```

```
Running Echo03 ../samples/slideSample01.xml


START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "Sample Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Wake up to WonderWidgets!
            END_ELM: </title>
        CHARS:
        END_ELM: </slide>
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
        CHARS:
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:    WonderWidgets
            END_ELM: </item>
        CHARS:
        END_ELM: </slide>
    CHARS:
    END_ELM: </slideshow>
END DOCUMENT
```

```
/*
 * @(#)Echo04.java 1.5 99/02/09
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo04 extends DefaultHandler
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        DefaultHandler handler = new Echo04();
        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler);

        } catch (Throwable t) {
            t.printStackTrace();
        }
        System.exit(0);
    }

    static private Writer  out;
    private String indentString = "    "; // Amount to indent
    private int indentLevel = 0;

    //===========================================================
    // SAX DocumentHandler methods
    //===========================================================

    public void setDocumentLocator(Locator l)
    {
        // Save this to resolve relative URIs or to give diagnostics.
        try {
          out.write("LOCATOR");
          out.write("\n SYS ID: " + l.getSystemId() );
          out.flush();
        } catch (IOException e) {
            // Ignore errors
        }
    }

    public void startDocument()
    throws SAXException
    {
        nl();
        nl();
        emit("START DOCUMENT");
        nl();
        emit("<?xml version='1.0' encoding='UTF-8'?>");
    }

    public void endDocument()
    throws SAXException
    {
        nl(); emit("END DOCUMENT");
        try {
            nl();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        indentLevel++;
        nl(); emit("ELEMENT: ");
        String eName = lName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                nl();
                emit("   ATTR: ");
                emit(aName);
                emit("\t\"");
                emit(attrs.getValue(i));
                emit("\"");
            }
        }
        if (attrs.getLength() > 0) nl();
        emit(">");
    }

    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          )
    throws SAXException
    {
        nl();
        emit("END_ELM: ");
        emit("</"+sName+">");
        indentLevel--;
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        nl(); emit("CHARS:   ");
        String s = new String(buf, offset, len);
        if (!s.trim().equals("")) emit(s);
    }

    //============================================================
    // Utility Methods ...
    //============================================================

    // Wrap I/O exceptions in SAX exceptions, to
    // suit handler signature requirements
    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    // Start a new line
    // and indent the next line appropriately
    private void nl()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
            for (int i=0; i < indentLevel; i++) out.write(indentString);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
```

```
}
```

```
Running Echo04 ../samples/slideSample01.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample01.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "Sample Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Wake up to WonderWidgets!
            END_ELM: </title>
        CHARS:
        END_ELM: </slide>
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
        CHARS:
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:    WonderWidgets
            END_ELM: </item>
        CHARS:
        END_ELM: </slide>
    CHARS:
    END_ELM: </slideshow>
END DOCUMENT
```

Why *WonderWidgets* are great Who *buys* WonderWidgets

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

<!--  A SAMPLE set of slides  -->

<slideshow
    title="Sample Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- PROCESSING INSTRUCTION -->
    <?my.presentation.Program QUERY="exec, tech, all"?>

    <!-- TITLE SLIDE -->
    <slide type="all">
       <title>Wake up to WonderWidgets!</title>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <title>Overview</title>
      <item>Why <em>WonderWidgets</em> are great</item>
      <item/>
      <item>Who <em>buys</em> WonderWidgets</item>
    </slide>
</slideshow>
```

```
/*
 * @(#)Echo05.java 1.5 99/02/09
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo05 extends DefaultHandler
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        DefaultHandler handler = new Echo05();
        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler);

        } catch (Throwable t) {
            t.printStackTrace();
        }
        System.exit(0);
    }

    static private Writer  out;
    private String indentString = "    "; // Amount to indent
    private int indentLevel = 0;

    //===========================================================
    // SAX DocumentHandler methods
    //===========================================================

    public void setDocumentLocator(Locator l)
    {
        // Save this to resolve relative URIs or to give diagnostics.
        try {
          out.write("LOCATOR");
          out.write("\n SYS ID: " + l.getSystemId() );
          out.flush();
        } catch (IOException e) {
            // Ignore errors
        }
    }

    public void startDocument()
    throws SAXException
    {
        nl();
        nl();
        emit("START DOCUMENT");
        nl();
        emit("<?xml version='1.0' encoding='UTF-8'?>");
    }

    public void endDocument()
    throws SAXException
    {
        nl(); emit("END DOCUMENT");
        try {
            nl();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        indentLevel++;
        nl(); emit("ELEMENT: ");
        String eName = lName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                nl();
                emit("   ATTR: ");
                emit(aName);
                emit("\t\"");
                emit(attrs.getValue(i));
                emit("\"");
            }
        }
        if (attrs.getLength() > 0) nl();
        emit(">");
    }

    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          )
    throws SAXException
    {
        nl();
        emit("END_ELM: ");
        emit("</"+sName+">");
        indentLevel--;
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        nl(); emit("CHARS:   ");
        String s = new String(buf, offset, len);
        if (!s.trim().equals("")) emit(s);
    }

    public void processingInstruction(String target, String data)
    throws SAXException
    {
        nl();
        emit("PROCESS: ");
        emit("<?"+target+" "+data+"?>");
    }

    //===========================================================
    // Utility Methods ...
    //===========================================================

    // Wrap I/O exceptions in SAX exceptions, to
    // suit handler signature requirements
    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    // Start a new line
    // and indent the next line appropriately
    private void nl()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
            for (int i=0; i < indentLevel; i++) out.write(indentString);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
}
```

```
Running Echo05 ../samples/slideSample02.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample02.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "Sample Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    CHARS:
    CHARS:
    PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Wake up to WonderWidgets!
            END_ELM: </title>
        CHARS:
        END_ELM: </slide>
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
        CHARS:
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:    WonderWidgets
            END_ELM: </item>
        CHARS:
        END_ELM: </slide>
    CHARS:
    END_ELM: </slideshow>
END DOCUMENT
```

Why *WonderWidgets* are great Who *buys* WonderWidgets

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

<!-- Slides with a fatal error -->

<slideshow
    title="Sample Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- TITLE SLIDE -->
    <slide type="all">
      <title>Wake up to WonderWidgets!</title>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <title>Overview</title>
      <item>Why <em>WonderWidgets</em> are great</item>
      <item>
      <item>Who <em>buys</em> WonderWidgets</item>
    </slide>

</slideshow>
```

```
Running Echo05 ../samples/slideSampleBad1.xml
LOCATOR
 SYS ID:
file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSampleBad1.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "Sample Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Wake up to WonderWidgets!
            END_ELM: </title>
        CHARS:
        END_ELM: </slide>
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
        CHARS:
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:
                ELEMENT: <item>
                CHARS:   Who
                    ELEMENT: <em>
                    CHARS:   buys
                    END_ELM: </em>
                CHARS:    WonderWidgets
                END_ELM: </item>
            CHARS:   org.xml.sax.SAXParseException: Expected "</item>" to terminate
```

```
element starting on line 20.
        at org.apache.crimson.parser.Parser2.fatal(Parser2.java:3035)
        at org.apache.crimson.parser.Parser2.fatal(Parser2.java:3029)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1474)
        at org.apache.crimson.parser.Parser2.content(Parser2.java:1700)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1468)
        at org.apache.crimson.parser.Parser2.content(Parser2.java:1700)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1468)
        at org.apache.crimson.parser.Parser2.parseInternal(Parser2.java:499)
        at org.apache.crimson.parser.Parser2.parse(Parser2.java:304)
        at org.apache.crimson.parser.XMLReaderImpl.parse(XMLReaderImpl.java:433)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:346)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:286)
        at Echo05.main(Echo05.java:60)
```

```
/*
 * @(#)Echo06.java 1.5 99/02/09
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo06 extends DefaultHandler
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        DefaultHandler handler = new Echo06();
        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler);

        } catch (SAXParseException spe) {
           // Error generated by the parser
```

```
            System.out.println("\n** Parsing error"
               + ", line " + spe.getLineNumber()
               + ", uri " + spe.getSystemId());
            System.out.println("   " + spe.getMessage() );

            // Use the contained exception, if any
            Exception  x = spe;
            if (spe.getException() != null)
                x = spe.getException();
            x.printStackTrace();

        } catch (SAXException sxe) {
            // Error generated by this application
            // (or a parser-initialization error)
            Exception  x = sxe;
            if (sxe.getException() != null)
                x = sxe.getException();
            x.printStackTrace();

        } catch (ParserConfigurationException pce) {
             // Parser with specified options can't be built
             pce.printStackTrace();

        } catch (IOException ioe) {
            // I/O error
            ioe.printStackTrace();
        }

        System.exit(0);
    }

    static private Writer  out;
    private String indentString = "    "; // Amount to indent
    private int indentLevel = 0;

    //===========================================================
    // SAX DocumentHandler methods
    //===========================================================

    public void setDocumentLocator(Locator l)
    {
        // Save this to resolve relative URIs or to give diagnostics.
        try {
          out.write("LOCATOR");
          out.write("\n SYS ID: " + l.getSystemId() );
          out.flush();
        } catch (IOException e) {
            // Ignore errors
        }
    }

    public void startDocument()
    throws SAXException
    {
        nl();
        nl();
        emit("START DOCUMENT");
        nl();
        emit("<?xml version='1.0' encoding='UTF-8'?>");
    }

    public void endDocument()
    throws SAXException
    {
        nl(); emit("END DOCUMENT");
        try {
            nl();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        indentLevel++;
        nl(); emit("ELEMENT: ");
        String eName = lName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                nl();
                emit("   ATTR: ");
                emit(aName);
                emit("\t\"");
                emit(attrs.getValue(i));
                emit("\"");
            }
        }
        if (attrs.getLength() > 0) nl();
        emit(">");
    }

    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          )
    throws SAXException
    {
        nl();
        emit("END_ELM: ");
        emit("</"+sName+">");
        indentLevel--;
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        nl(); emit("CHARS:   ");
        String s = new String(buf, offset, len);
        if (!s.trim().equals("")) emit(s);
    }

    public void processingInstruction(String target, String data)
    throws SAXException
    {
        nl();
        emit("PROCESS: ");
        emit("<?"+target+" "+data+"?>");
    }

    //===========================================================
    // Utility Methods ...
    //===========================================================

    // Wrap I/O exceptions in SAX exceptions, to
    // suit handler signature requirements
    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    // Start a new line
    // and indent the next line appropriately
    private void nl()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
            for (int i=0; i < indentLevel; i++) out.write(indentString);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
}
```

```
Running Echo06 ../samples/slideSampleBad1.xml
LOCATOR
 SYS ID:
file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSampleBad1.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "Sample Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Wake up to WonderWidgets!
            END_ELM: </title>
        CHARS:
        END_ELM: </slide>
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
        CHARS:
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:
                ELEMENT: <item>
                CHARS:   Who
                    ELEMENT: <em>
                    CHARS:   buys
                    END_ELM: </em>
                CHARS:    WonderWidgets
                END_ELM: </item>
            CHARS:
** Parsing error, line 22, uri
file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSampleBad1.xml
   Expected "</item>" to terminate element starting on line 20.
org.xml.sax.SAXParseException: Expected "</item>" to terminate element starting on
line 20.
        at org.apache.crimson.parser.Parser2.fatal(Parser2.java:3035)
        at org.apache.crimson.parser.Parser2.fatal(Parser2.java:3029)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1474)
        at org.apache.crimson.parser.Parser2.content(Parser2.java:1700)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1468)
        at org.apache.crimson.parser.Parser2.content(Parser2.java:1700)
        at org.apache.crimson.parser.Parser2.maybeElement(Parser2.java:1468)
        at org.apache.crimson.parser.Parser2.parseInternal(Parser2.java:499)
        at org.apache.crimson.parser.Parser2.parse(Parser2.java:304)
        at org.apache.crimson.parser.XMLReaderImpl.parse(XMLReaderImpl.java:433)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:346)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:286)
        at Echo06.main(Echo06.java:60)
```

Why *WonderWidgets* are great Who *buys* WonderWidgets

```xml
<?xml version='1.2' encoding='utf-8'?>

<!-- Slides with a non-fatal error -->

<slideshow
    title="Sample Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- TITLE SLIDE -->
    <slide type="all">
      <title>Wake up to WonderWidgets!</title>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <title>Overview</title>
      <item>Why <em>WonderWidgets</em> are great</item>
      <item/>
      <item>Who <em>buys</em> WonderWidgets</item>
    </slide>

</slideshow>
```

```
Running Echo06 ../samples/slideSampleBad2.xml
LOCATOR
 SYS ID:
file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSampleBad2.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "Sample Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Wake up to WonderWidgets!
            END_ELM: </title>
        CHARS:
        END_ELM: </slide>
    CHARS:
    CHARS:
        ELEMENT: <slide
           ATTR: type   "all"
        >
        CHARS:
            ELEMENT: <title>
            CHARS:   Overview
            END_ELM: </title>
        CHARS:
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            END_ELM: </item>
        CHARS:
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:    WonderWidgets
            END_ELM: </item>
        CHARS:
        END_ELM: </slide>
    CHARS:
    END_ELM: </slideshow>
END DOCUMENT
```

```
/*
 * @(#)Echo07.java 1.5 99/02/09
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class Echo07 extends DefaultHandler
{
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        DefaultHandler handler = new Echo07();
        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler);

        } catch (SAXParseException spe) {
           // Error generated by the parser
           System.out.println("\n** Parsing error"
              + ", line " + spe.getLineNumber()
              + ", uri " + spe.getSystemId());
           System.out.println("   " + spe.getMessage() );

           // Use the contained exception, if any
           Exception  x = spe;
           if (spe.getException() != null)
               x = spe.getException();
           x.printStackTrace();

        } catch (SAXException sxe) {
           // Error generated by this application
           // (or a parser-initialization error)
           Exception  x = sxe;
           if (sxe.getException() != null)
               x = sxe.getException();
           x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }

        System.exit(0);
    }

    static private Writer  out;
    private String indentString = "    "; // Amount to indent
    private int indentLevel = 0;

    //===========================================================
    // SAX DocumentHandler methods
    //===========================================================

    public void setDocumentLocator(Locator l)
    {
        // Save this to resolve relative URIs or to give diagnostics.
        try {
          out.write("LOCATOR");
          out.write("\n SYS ID: " + l.getSystemId() );
          out.flush();
        } catch (IOException e) {
            // Ignore errors
        }
    }

    public void startDocument()
    throws SAXException
    {
        nl();
        nl();
        emit("START DOCUMENT");
        nl();
        emit("<?xml version='1.0' encoding='UTF-8'?>");
    }

    public void endDocument()
```

```
    throws SAXException
    {
        nl(); emit("END DOCUMENT");
        try {
            nl();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String namespaceURI,
                             String lName, // local name
                             String qName, // qualified name
                             Attributes attrs)
    throws SAXException
    {
        indentLevel++;
        nl(); emit("ELEMENT: ");
        String eName = lName; // element name
        if ("".equals(eName)) eName = qName; // namespaceAware = false
        emit("<"+eName);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                nl();
                emit("   ATTR: ");
                emit(aName);
                emit("\t\"");
                emit(attrs.getValue(i));
                emit("\"");
            }
        }
        if (attrs.getLength() > 0) nl();
        emit(">");
    }

    public void endElement(String namespaceURI,
                           String sName, // simple name
                           String qName  // qualified name
                          )
    throws SAXException
    {
        nl();
        emit("END_ELM: ");
        emit("</"+sName+">");
        indentLevel--;
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        nl(); emit("CHARS:   ");
        String s = new String(buf, offset, len);
        if (!s.trim().equals("")) emit(s);
    }

    public void processingInstruction(String target, String data)
    throws SAXException
    {
        nl();
        emit("PROCESS: ");
        emit("<?"+target+" "+data+"?>");
    }

    //===========================================================
    // SAX ErrorHandler methods
    //===========================================================

    // treat validation errors as fatal
    public void error(SAXParseException e)
    throws SAXParseException
    {
        throw e;
    }

    // dump warnings too
    public void warning(SAXParseException err)
    throws SAXParseException
    {
        System.out.println("** Warning"
            + ", line " + err.getLineNumber()
            + ", uri " + err.getSystemId());
        System.out.println("   " + err.getMessage());
    }

    //===========================================================
    // Utility Methods ...
    //===========================================================

    // Wrap I/O exceptions in SAX exceptions, to
    // suit handler signature requirements
    private void emit(String s)
    throws SAXException
    {
        try {
            out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    // Start a new line
    // and indent the next line appropriately
    private void nl()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
            for (int i=0; i < indentLevel; i++) out.write(indentString);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
}
```

```
Running Echo07 ../samples/slideSampleBad2.xml
LOCATOR
 SYS ID:
file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSampleBad2.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
** Parsing error, line 1, uri
file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSampleBad2.xml
   XML version "1.0" is recognized, but not "1.2".
org.xml.sax.SAXParseException: XML version "1.0" is recognized, but not "1.2".
        at org.apache.crimson.parser.Parser2.error(Parser2.java:3013)
        at org.apache.crimson.parser.Parser2.readVersion(Parser2.java:1070)
        at org.apache.crimson.parser.Parser2.maybeXmlDecl(Parser2.java:1002)
        at org.apache.crimson.parser.Parser2.parseInternal(Parser2.java:485)
        at org.apache.crimson.parser.Parser2.parse(Parser2.java:304)
        at org.apache.crimson.parser.XMLReaderImpl.parse(XMLReaderImpl.java:433)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:346)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:286)
        at Echo07.main(Echo07.java:60)
```

```
Running Echo10 ../samples/slideSample08.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample08.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
** Parsing error, line 22, uri
file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideshow2.dtd
   Element "title" was already declared.
org.xml.sax.SAXParseException: Element "title" was already declared.
        at org.apache.crimson.parser.Parser2.error(Parser2.java:3013)
        at org.apache.crimson.parser.Parser2.maybeElementDecl(Parser2.java:1781)
        at org.apache.crimson.parser.Parser2.maybeMarkupDecl(Parser2.java:1196)
        at
org.apache.crimson.parser.Parser2.externalParameterEntity(Parser2.java:2751)
        at org.apache.crimson.parser.Parser2.maybeDoctypeDecl(Parser2.java:1154)
        at org.apache.crimson.parser.Parser2.parseInternal(Parser2.java:488)
        at org.apache.crimson.parser.Parser2.parse(Parser2.java:304)
        at org.apache.crimson.parser.XMLReaderImpl.parse(XMLReaderImpl.java:433)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:346)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java:286)
        at Echo10.main(Echo10.java:62)
```

%xhtml;

```
<?xml version='1.0' encoding='us-ascii'?>

<!--
    DTD for a simple "slide show".
-->

<!ELEMENT slideshow (slide+)>
<!ATTLIST slideshow
            title    CDATA    #REQUIRED
            date     CDATA    #IMPLIED
            author   CDATA    "unknown"
>
<!ELEMENT slide (image?, slide-title?, item*)>
<!ATTLIST slide
            type   (tech | exec | all) #IMPLIED
>

<!-- Defines the %inline; declaration -->
<!ENTITY % xhtml SYSTEM "xhtml.dtd">
%xhtml;

<!ELEMENT slide-title (%inline;)*>
<!ELEMENT item (%inline; | item)* >
<!ELEMENT image EMPTY>
<!ATTLIST image
            alt    CDATA    #IMPLIED
            src    CDATA    #REQUIRED
            type   CDATA    "image/gif"
>
```

]> Wake up to &products;! ©right; Overview Why *&products;* are great Who *buys* &products;

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

<!--  A SAMPLE set of slides  -->

<!DOCTYPE slideshow SYSTEM "slideshow3.dtd" [
  <!ENTITY product  "WonderWidget">
  <!ENTITY products "WonderWidgets">   <!-- FOR WALLY / WALLIES -->
  <!ENTITY copyright SYSTEM "copyright.xml">
]>

<!-- SUBSTITUTIONS WORK IN ATTRIBUTES, TOO -->
<slideshow
    title="&product; Slide Show"
    date="Date of publication"
    author="Yours Truly"
    >

    <!-- PROCESSING INSTRUCTION -->
    <?my.presentation.Program QUERY="exec, tech, all"?>

    <!-- TITLE SLIDE -->
    <slide type="all">
       <slide-title>Wake up to &products;!</slide-title>
    </slide>

    <!-- TITLE SLIDE -->
    <slide type="all">
       <item>&copyright;</item>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <slide-title>Overview</slide-title>
      <item>Why <em>&products;</em> are great</item>
      <item/>
      <item>Who <em>buys</em> &products;</item>
    </slide>

</slideshow>
```

```
<?xml version='1.0' encoding='us-ascii'?>

<!--
    This DTD does some of what the W3C is getting ready to do with its
    "XHTML" work (nee "Voyager").  It differs from the current WG draft
    because it uses namespaces correctly (!), and it isn't as complete
    even for HTML 3.2 support (much less 4.0) or, probably, correct.

    Note that what XHTML needs to do is become modular enough that XHTML
    can be used as a mixin with other document types, including either
    "the whole megillah" or just selected modules (e.g. omitting tables).
    That must work both ways ... other things as mixins to XHTML, and
    XHTML as a mixin to other things.

    THIS WILL BE REPLACED WITH A BETTER DTD AT SOME POINT.
-->

<!ELEMENT html (head, body)>
<!ATTLIST html
    xmlns       CDATA   #FIXED "http://www.example.com/xhtml"
    >

<!ELEMENT head (title,style*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT style (#PCDATA)>
<!ATTLIST style
    type        CDATA   #IMPLIED
    >

<!ENTITY % content "p|h1|h2|h3|h4|h5|h6|ul|ol|table|center">

<!ENTITY % inline "#PCDATA|em|b|a|img|br">
<!ELEMENT em (#PCDATA|a|b|img|br)*>
<!ELEMENT b (#PCDATA|a|em|img|br)*>
<!ELEMENT a (#PCDATA|b|em|img|br)*>
<!ATTLIST a
    href        CDATA   #IMPLIED
    name        CDATA   #IMPLIED
    >
<!ELEMENT br EMPTY>
<!ELEMENT img EMPTY>
<!ATTLIST img
    alt         CDATA   #IMPLIED
    border      CDATA   #IMPLIED
    height      CDATA   #IMPLIED
    src         CDATA   #REQUIRED
    width       CDATA   #IMPLIED
    >

<!ELEMENT body (%content;)+>
<!ATTLIST body
    bgcolor CDATA       #IMPLIED
    >

<!ELEMENT p (%inline;)*>
<!ELEMENT h1 (%inline;)*>
<!ELEMENT h2 (%inline;)*>
<!ELEMENT h3 (%inline;)*>
<!ELEMENT h4 (%inline;)*>
<!ELEMENT h5 (%inline;)*>
<!ELEMENT h6 (%inline;)*>

<!ELEMENT ul (li+)>
<!ELEMENT ol (li+)>
<!ELEMENT li (%inline;)*>

<!ELEMENT table (tr+)>
<!ATTLIST table
    height      CDATA                   #IMPLIED
    width       CDATA                   #IMPLIED
    align       (left|center|right)     #IMPLIED
    cellspacing CDATA                   #IMPLIED
    >
<!ELEMENT tr (td+)>
<!ATTLIST tr
    align       (left|center|right)             #IMPLIED
    valign      (top|center|bottom|baseline)    #IMPLIED
    >
<!ELEMENT td (%inline;|%content;)*>
<!ATTLIST td
    height      CDATA                           #IMPLIED
    width       CDATA                           #IMPLIED
    align       (left|center|right)             #IMPLIED
    valign      (top|center|bottom|baseline)    #IMPLIED
    rowspan     CDATA                           #IMPLIED
    colspan     CDATA                           #IMPLIED
    >

<!ELEMENT center (%inline;|%content;)*>
```

```
Running Echo10 ../samples/slideSample09.xml
LOCATOR
 SYS ID: file:/java/pubs/dev/xml/docs/tutorial/sax/work/../samples/slideSample09.xml

START DOCUMENT
<?xml version='1.0' encoding='UTF-8'?>
    ELEMENT: <slideshow
       ATTR: title      "WonderWidget Slide Show"
       ATTR: date       "Date of publication"
       ATTR: author     "Yours Truly"
    >
    PROCESS: <?my.presentation.Program QUERY="exec, tech, all"?>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <slide-title>
            CHARS:   Wake up to
            CHARS:   WonderWidgets
            CHARS:   !
            END_ELM: </slide-title>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <item>
            CHARS:
This is the standard copyright message that our lawyers
make us put everywhere so we don't have to shell out a
million bucks every time someone spills hot coffee in their
lap...

            END_ELM: </item>
        END_ELM: </slide>
        ELEMENT: <slide
           ATTR: type   "all"
        >
            ELEMENT: <slide-title>
            CHARS:   Overview
            END_ELM: </slide-title>
            ELEMENT: <item>
            CHARS:   Why
                ELEMENT: <em>
                CHARS:   WonderWidgets
                END_ELM: </em>
            CHARS:    are great
            END_ELM: </item>
            ELEMENT: <item>
            END_ELM: </item>
            ELEMENT: <item>
            CHARS:   Who
                ELEMENT: <em>
                CHARS:   buys
                END_ELM: </em>
            CHARS:
            CHARS:   WonderWidgets
            END_ELM: </item>
        END_ELM: </slide>
    END_ELM: </slideshow>
END DOCUMENT
```

```
/*
 * @(#)DomEcho01.java   1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
import java.io.*;

public class TransformationApp01
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

    public static void main (String argv [])
    {
        if (argv.length != 1) {
            System.err.println ("Usage: java TransformationApp filename");
            System.exit (1);
        }

        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        //factory.setNamespaceAware(true);
        //factory.setValidating(true);

        try {
            File f = new File(argv[0]);

            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(f);
         } catch (SAXException sxe) {
           // Error generated by this application
           // (or a parser-initialization error)
           Exception  x = sxe;
           if (sxe.getException() != null)
               x = sxe.getException();
           x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }

    } // main

}
```

```
/*
 * @(#)TransformationApp02.java 1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

// For write operation
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import java.io.*;

public class TransformationApp02
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

    public static void main (String argv [])
    {
        if (argv.length != 1) {
            System.err.println ("Usage: java TransformationApp filename");
            System.exit (1);
        }
        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        //factory.setNamespaceAware(true);
        //factory.setValidating(true);

        try {
            File f = new File(argv[0]);

            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(f);

            // Use a Transformer for output
            TransformerFactory tFactory =
                TransformerFactory.newInstance();
            Transformer transformer = tFactory.newTransformer();

            DOMSource source = new DOMSource(document);
            StreamResult result = new StreamResult(System.out);
            transformer.transform(source, result);

        } catch (TransformerConfigurationException tce) {
           // Error generated by the parser
           System.out.println ("\n** Transformer Factory error");
           System.out.println("   " + tce.getMessage() );

           // Use the contained exception, if any
           Throwable x = tce;
           if (tce.getException() != null)
               x = tce.getException();
           x.printStackTrace();

        } catch (TransformerException te) {
           // Error generated by the parser
           System.out.println ("\n** Transformation error");
           System.out.println("   " + te.getMessage() );

           // Use the contained exception, if any
           Throwable x = te;
           if (te.getException() != null)
               x = te.getException();
           x.printStackTrace();

         } catch (SAXException sxe) {
           // Error generated by this application
           // (or a parser-initialization error)
           Exception  x = sxe;
           if (sxe.getException() != null)
               x = sxe.getException();
           x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }

    } // main

}
```

```
/*
 * @(#)DomEcho03.java   1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

// For write operation
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import java.io.*;

public class TransformationApp03
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

    public static void main (String argv [])
    {
        if (argv.length != 1) {
            System.err.println ("Usage: java TransformationApp filename");
            System.exit (1);
        }

        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        //factory.setNamespaceAware(true);
        //factory.setValidating(true);

        try {
            File f = new File(argv[0]);

            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(f);


            // Get the first <slide> element in the DOM
            NodeList list = document.getElementsByTagName("slide");
            Node node = list.item(0);

            // Use a Transformer for output
            TransformerFactory tFactory =
                TransformerFactory.newInstance();
            Transformer transformer = tFactory.newTransformer();
            DOMSource source = new DOMSource(node);
            StreamResult result = new StreamResult(System.out);
            transformer.transform(source, result);

        } catch (TransformerConfigurationException tce) {
           // Error generated by the parser
           System.out.println ("\n** Transformer Factory error");
           System.out.println("   " + tce.getMessage() );

           // Use the contained exception, if any
           Throwable x = tce;
           if (tce.getException() != null)
               x = tce.getException();
           x.printStackTrace();

        } catch (TransformerException te) {
           // Error generated by the parser
           System.out.println ("\n** Transformation error");
           System.out.println("   " + te.getMessage() );

           // Use the contained exception, if any
           Throwable x = te;
           if (te.getException() != null)
               x = te.getException();
           x.printStackTrace();

         } catch (SAXException sxe) {
           // Error generated by this application
           // (or a parser-initialization error)
           Exception  x = sxe;
           if (sxe.getException() != null)
               x = sxe.getException();
           x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }

    } // main

}
```

```
Running TransformationApp02 ../samples/slideSample01.xml
<?xml version="1.0" encoding="UTF-8"?>
<!--  A SAMPLE set of slides  --><slideshow title="Sample Slide Show" date="Date of
publication" author="Yours Truly">

    <!-- TITLE SLIDE -->
    <slide type="all">
      <title>Wake up to WonderWidgets!</title>
    </slide>

    <!-- OVERVIEW -->
    <slide type="all">
      <title>Overview</title>
      <item>Why <em>WonderWidgets</em> are great</item>
      <item/>
      <item>Who <em>buys</em> WonderWidgets</item>
    </slide>
```

```
Running TransformationApp03 ../samples/slideSample01.xml
<?xml version="1.0" encoding="UTF-8"?>
<slide type="all">
      <title>Wake up to WonderWidgets!</title>
```

```
/*
 * @(#)DomEcho04.java   1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.ContentHandler;
import org.xml.sax.InputSource;

// For write operation
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.sax.SAXSource;
import javax.xml.transform.stream.StreamResult;

import java.io.*;

public class TransformationApp04
{

    public static void main (String argv [])
    {
        if (argv.length != 1) {
            System.err.println ("Usage: java TransformationApp filename");
            System.exit (1);
        }


        // Create the sax "parser".
        AddressBookReader02 saxReader = new AddressBookReader02();

        try {
            File f = new File(argv[0]);

            // Use a Transformer for output
            TransformerFactory tFactory =
                TransformerFactory.newInstance();
            Transformer transformer = tFactory.newTransformer();

            // Use the parser as a SAX source for input
            FileReader fr = new FileReader(f);
            BufferedReader br = new BufferedReader(fr);
            InputSource inputSource = new InputSource(fr);
            SAXSource source = new SAXSource(saxReader, inputSource);
            StreamResult result = new StreamResult(System.out);
            transformer.transform(source, result);

        } catch (TransformerConfigurationException tce) {
           // Error generated by the parser
           System.out.println ("\n** Transformer Factory error");
           System.out.println("   " + tce.getMessage() );

           // Use the contained exception, if any
           Throwable x = tce;
           if (tce.getException() != null)
               x = tce.getException();
           x.printStackTrace();

        } catch (TransformerException te) {
           // Error generated by the parser
           System.out.println ("\n** Transformation error");
           System.out.println("   " + te.getMessage() );

           // Use the contained exception, if any
           Throwable x = te;
           if (te.getException() != null)
               x = te.getException();
           x.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }

    } // main

}
```

```
/*
 * @(#)DomEcho01.java   1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import java.io.File;
import java.io.IOException;

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

public class DomEcho01{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: java DomEcho filename");
            System.exit(1);
        }

        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        //factory.setValidating(true);
        //factory.setNamespaceAware(true);
        try {
           DocumentBuilder builder = factory.newDocumentBuilder();
           document = builder.parse( new File(argv[0]) );
        } catch (SAXException sxe) {
           // Error generated during parsing)
           Exception  x = sxe;
           if (sxe.getException() != null)
               x = sxe.getException();
           x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }
    } // main

}
```

```
/*
 * @(#)DomEcho02.java   1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import java.io.File;
import java.io.IOException;

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

// Basic GUI components
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;

// GUI components for right-hand side
import javax.swing.JSplitPane;
import javax.swing.JEditorPane;

// GUI support classes
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.WindowEvent;
import java.awt.event.WindowAdapter;

// For creating borders
```

```
import javax.swing.border.EmptyBorder;
import javax.swing.border.BevelBorder;
import javax.swing.border.CompoundBorder;

// For creating a TreeModel
import javax.swing.tree.*;
import javax.swing.event.*;
import java.util.*;

public class DomEcho02  extends JPanel
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

    static final int windowHeight = 460;
    static final int leftWidth = 300;
    static final int rightWidth = 340;
    static final int windowWidth = leftWidth + rightWidth;

    public DomEcho02()
    {
       // Make a nice border
       EmptyBorder eb = new EmptyBorder(5,5,5,5);
       BevelBorder bb = new BevelBorder(BevelBorder.LOWERED);
       CompoundBorder cb = new CompoundBorder(eb,bb);
       this.setBorder(new CompoundBorder(cb,eb));

       // Set up the tree
       JTree tree = new JTree(new DomToTreeModelAdapter());

       // Iterate over the tree and make nodes visible
       // (Otherwise, the tree shows up fully collapsed)
       //TreePath nodePath = ???;
       //  tree.expandPath(nodePath);

       // Build left-side view
       JScrollPane treeView = new JScrollPane(tree);
       treeView.setPreferredSize(
           new Dimension( leftWidth, windowHeight ));

       // Build right-side view
       JEditorPane htmlPane = new JEditorPane("text/html","");
       htmlPane.setEditable(false);
       JScrollPane htmlView = new JScrollPane(htmlPane);
       htmlView.setPreferredSize(
           new Dimension( rightWidth, windowHeight ));

       // Build split-pane view
       JSplitPane splitPane =
          new JSplitPane( JSplitPane.HORIZONTAL_SPLIT,
                          treeView,
                          htmlView );
       splitPane.setContinuousLayout( true );
       splitPane.setDividerLocation( leftWidth );
       splitPane.setPreferredSize(
            new Dimension( windowWidth + 10, windowHeight+10 ));

       // Add GUI components
       this.setLayout(new BorderLayout());
       this.add("Center", splitPane );
    } // constructor
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: java DomEcho filename");
            System.exit(1);
        }

        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        //factory.setValidating(true);
        //factory.setNamespaceAware(true);
        try {
           DocumentBuilder builder = factory.newDocumentBuilder();
           document = builder.parse( new File(argv[0]) );
            makeFrame();

        } catch (SAXException sxe) {
           // Error generated during parsing)
           Exception  x = sxe;
           if (sxe.getException() != null)
               x = sxe.getException();
           x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }
    } // main

    public static void makeFrame() {
        // Set up a GUI framework
        JFrame frame = new JFrame("DOM Echo");
        frame.addWindowListener(
          new WindowAdapter() {
            public void windowClosing(WindowEvent e) {System.exit(0);}
          }
        );

        // Set up the tree, the views, and display it all
        final DomEcho02 echoPanel =
           new DomEcho02();
        frame.getContentPane().add("Center", echoPanel );
        frame.pack();
        Dimension screenSize =
           Toolkit.getDefaultToolkit().getScreenSize();
        int w = windowWidth + 10;
        int h = windowHeight + 10;
        frame.setLocation(screenSize.width/3 - w/2,
                          screenSize.height/2 - h/2);
        frame.setSize(w, h);
        frame.setVisible(true);
    } // makeFrame


    // An array of names for DOM node-types
    // (Array indexes = nodeType() values.)
    static final String[] typeName = {
        "none",
        "Element",
        "Attr",
        "Text",
        "CDATA",
        "EntityRef",
        "Entity",
        "ProcInstr",
        "Comment",
        "Document",
        "DocType",
        "DocFragment",
        "Notation",
    };

    // This class wraps a DOM node and returns the text we want to
    // display in the tree. It also returns children, index values,
    // and child counts.
    public class AdapterNode
    {
      org.w3c.dom.Node domNode;

      // Construct an Adapter node from a DOM node
      public AdapterNode(org.w3c.dom.Node node) {
        domNode = node;
      }

      // Return a string that identifies this node in the tree
      // *** Refer to table at top of org.w3c.dom.Node ***
      public String toString() {
        String s = typeName[domNode.getNodeType()];
        String nodeName = domNode.getNodeName();
        if (! nodeName.startsWith("#")) {
           s += ": " + nodeName;
        }
        if (domNode.getNodeValue() != null) {
           if (s.startsWith("ProcInstr"))
              s += ", ";
           else
              s += ": ";
           // Trim the value to get rid of NL's at the front
           String t = domNode.getNodeValue().trim();
           int x = t.indexOf("\n");
           if (x >= 0) t = t.substring(0, x);
           s += t;
        }
        return s;
      }


      /*
       * Return children, index, and count values
       */
      public int index(AdapterNode child) {
        //System.err.println("Looking for index of " + child);
        int count = childCount();
        for (int i=0; i<count; i++) {
          AdapterNode n = this.child(i);
          if (child.domNode == n.domNode) return i;
        }
        return -1; // Should never get here.
      }
      public AdapterNode child(int searchIndex) {
        //Note: JTree index is zero-based.
        org.w3c.dom.Node node =
             domNode.getChildNodes().item(searchIndex);
        return new AdapterNode(node);
      }

      public int childCount() {
          return domNode.getChildNodes().getLength();
      }
    }

    // This adapter converts the current Document (a DOM) into
    // a JTree model.
    public class DomToTreeModelAdapter
      implements javax.swing.tree.TreeModel
    {
      // Basic TreeModel operations
      public Object  getRoot() {
        //System.err.println("Returning root: " +document);
        return new AdapterNode(document);
      }
      public boolean isLeaf(Object aNode) {
        // Determines whether the icon shows up to the left.
        // Return true for any node with no children
        AdapterNode node = (AdapterNode) aNode;
        if (node.childCount() > 0) return false;
        return true;
      }
      public int     getChildCount(Object parent) {
        AdapterNode node = (AdapterNode) parent;
        return node.childCount();
      }
      public Object getChild(Object parent, int index) {
        AdapterNode node = (AdapterNode) parent;
        return node.child(index);
      }
      public int getIndexOfChild(Object parent, Object child) {
        AdapterNode node = (AdapterNode) parent;
        return node.index((AdapterNode) child);
      }
      public void valueForPathChanged(TreePath path, Object newValue) {
        // Null. We won't be making changes in the GUI
        // If we did, we would ensure the new value was really new,
        // adjust the model, and then fire a TreeNodesChanged event.
      }

      /*
       * Use these methods to add and remove event listeners.
       * (Needed to satisfy TreeModel interface, but not used.)
       */
      private Vector listenerList = new Vector();
      public void addTreeModelListener(TreeModelListener listener) {
        if ( listener != null
        && ! listenerList.contains( listener ) ) {
           listenerList.addElement( listener );
        }
      }
      public void removeTreeModelListener(TreeModelListener listener) {
        if ( listener != null ) {
           listenerList.removeElement( listener );
        }
      }

      // Note: Since XML works with 1.1, this example uses Vector.
      // If coding for 1.2 or later, though, I'd use this instead:
      //   private List listenerList = new LinkedList();
      // The operations on the List are then add(), remove() and
      // iteration, via:
      //  Iterator it = listenerList.iterator();
      //  while ( it.hasNext() ) {
      //    TreeModelListener listener = (TreeModelListener) it.next();
      //    ...
      //  }

      /*
       * Invoke these methods to inform listeners of changes.
       * (Not needed for this example.)
       * Methods taken from TreeModelSupport class described at
       *   http://java.sun.com/products/jfc/tsc/articles/jtree/index.html
       * That architecture (produced by Tom Santos and Steve Wilson)
       * is more elegant. I just hacked 'em in here so they are
       * immediately at hand.
       */
      public void fireTreeNodesChanged( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
          TreeModelListener listener =
            (TreeModelListener) listeners.nextElement();
          listener.treeNodesChanged( e );
        }
      }
      public void fireTreeNodesInserted( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
           TreeModelListener listener =
             (TreeModelListener) listeners.nextElement();
           listener.treeNodesInserted( e );
        }
      }
      public void fireTreeNodesRemoved( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
          TreeModelListener listener =
            (TreeModelListener) listeners.nextElement();
          listener.treeNodesRemoved( e );
        }
      }
      public void fireTreeStructureChanged( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
          TreeModelListener listener =
            (TreeModelListener) listeners.nextElement();
          listener.treeStructureChanged( e );
        }
      }
    }
}
```

This is the standard copyright message that our lawyers make us put everywhere so we don't have to shell out a million bucks every time someone spills hot coffee in their lap...

```
<!--  A SAMPLE copyright  -->
This is the standard copyright message that our lawyers
make us put everywhere so we don't have to shell out a
million bucks every time someone spills hot coffee in their
lap...
```

```
/*
 * @(#)DomEcho03.java   1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import java.io.File;
import java.io.IOException;

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

// Basic GUI components
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;

// GUI components for right-hand side
import javax.swing.JSplitPane;
import javax.swing.JEditorPane;

// GUI support classes
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.WindowEvent;
import java.awt.event.WindowAdapter;

// For creating borders
```

```
import javax.swing.border.EmptyBorder;
import javax.swing.border.BevelBorder;
import javax.swing.border.CompoundBorder;

// For creating a TreeModel
import javax.swing.tree.*;
import javax.swing.event.*;
import java.util.*;

public class DomEcho03  extends JPanel
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

    boolean compress = true;
    static final int windowHeight = 460;
    static final int leftWidth = 300;
    static final int rightWidth = 340;
    static final int windowWidth = leftWidth + rightWidth;

    public DomEcho03()
    {
       // Make a nice border
       EmptyBorder eb = new EmptyBorder(5,5,5,5);
       BevelBorder bb = new BevelBorder(BevelBorder.LOWERED);
       CompoundBorder cb = new CompoundBorder(eb,bb);
       this.setBorder(new CompoundBorder(cb,eb));

       // Set up the tree
       JTree tree = new JTree(new DomToTreeModelAdapter());

       // Iterate over the tree and make nodes visible
       // (Otherwise, the tree shows up fully collapsed)
       //TreePath nodePath = ???;
       //  tree.expandPath(nodePath);

       // Build left-side view
       JScrollPane treeView = new JScrollPane(tree);
       treeView.setPreferredSize(
           new Dimension( leftWidth, windowHeight ));

       // Build right-side view
       JEditorPane htmlPane = new JEditorPane("text/html","");
       htmlPane.setEditable(false);
       JScrollPane htmlView = new JScrollPane(htmlPane);
       htmlView.setPreferredSize(
           new Dimension( rightWidth, windowHeight ));

       // Build split-pane view
       JSplitPane splitPane =
          new JSplitPane( JSplitPane.HORIZONTAL_SPLIT,
                          treeView,
                          htmlView );
       splitPane.setContinuousLayout( true );
       splitPane.setDividerLocation( leftWidth );
       splitPane.setPreferredSize(
            new Dimension( windowWidth + 10, windowHeight+10 ));

       // Add GUI components
       this.setLayout(new BorderLayout());
       this.add("Center", splitPane );
    } // constructor
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: java DomEcho filename");
            System.exit(1);
        }

        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        //factory.setValidating(true);
        //factory.setNamespaceAware(true);
        try {
           DocumentBuilder builder = factory.newDocumentBuilder();
           document = builder.parse( new File(argv[0]) );
            makeFrame();

        } catch (SAXException sxe) {
           // Error generated during parsing)
           Exception  x = sxe;
           if (sxe.getException() != null)
               x = sxe.getException();
           x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }
    } // main

    public static void makeFrame() {
        // Set up a GUI framework
        JFrame frame = new JFrame("DOM Echo");
        frame.addWindowListener(
          new WindowAdapter() {
            public void windowClosing(WindowEvent e) {System.exit(0);}
          }
        );

        // Set up the tree, the views, and display it all
        final DomEcho03 echoPanel =
           new DomEcho03();
        frame.getContentPane().add("Center", echoPanel );
        frame.pack();
        Dimension screenSize =
           Toolkit.getDefaultToolkit().getScreenSize();
        int w = windowWidth + 10;
        int h = windowHeight + 10;
        frame.setLocation(screenSize.width/3 - w/2,
                          screenSize.height/2 - h/2);
        frame.setSize(w, h);
        frame.setVisible(true);
    } // makeFrame


    // An array of names for DOM node-types
    // (Array indexes = nodeType() values.)
    static final String[] typeName = {
        "none",
        "Element",
        "Attr",
        "Text",
        "CDATA",
        "EntityRef",
        "Entity",
        "ProcInstr",
        "Comment",
        "Document",
        "DocType",
        "DocFragment",
        "Notation",
    };
    static final int ELEMENT_TYPE =   1;
    // The list of elements to display in the tree
    // Could set this with a command-line argument, but
    // not much point -- the list of tree elements still
    // has to be defined internally.
    // Extra credit: Read the list from a file
    // Super-extra credit: Process a DTD and build the list.
   static String[] treeElementNames = {
        "slideshow",
        "slide",
        "title",         // For slideshow #1
        "slide-title",   // For slideshow #10
        "item",
    };
    boolean treeElement(String elementName) {
      for (int i=0; i<treeElementNames.length; i++) {
        if ( elementName.equals(treeElementNames[i]) ) return true;
      }
      return false;
    }

    // This class wraps a DOM node and returns the text we want to
    // display in the tree. It also returns children, index values,
    // and child counts.
    public class AdapterNode
    {
      org.w3c.dom.Node domNode;

      // Construct an Adapter node from a DOM node
      public AdapterNode(org.w3c.dom.Node node) {
        domNode = node;
      }

      // Return a string that identifies this node in the tree
      // *** Refer to table at top of org.w3c.dom.Node ***
      public String toString() {
        String s = typeName[domNode.getNodeType()];
        String nodeName = domNode.getNodeName();
        if (! nodeName.startsWith("#")) {
           s += ": " + nodeName;
        }
        if (domNode.getNodeValue() != null) {
           if (s.startsWith("ProcInstr"))
              s += ", ";
           else
              s += ": ";
           // Trim the value to get rid of NL's at the front
```

```
          String t = domNode.getNodeValue().trim();
          int x = t.indexOf("\n");
          if (x >= 0) t = t.substring(0, x);
          s += t;
       }
       return s;
     }


     /*
      * Return children, index, and count values
      */
     public int index(AdapterNode child) {
       //System.err.println("Looking for index of " + child);
       int count = childCount();
       for (int i=0; i<count; i++) {
         AdapterNode n = this.child(i);
         if (child.domNode == n.domNode) return i;
       }
       return -1; // Should never get here.
     }

     public AdapterNode child(int searchIndex) {
       //Note: JTree index is zero-based.
       org.w3c.dom.Node node =
            domNode.getChildNodes().item(searchIndex);
       if (compress) {
         // Return Nth displayable node
         int elementNodeIndex = 0;
         for (int i=0; i<domNode.getChildNodes().getLength(); i++) {
           node = domNode.getChildNodes().item(i);
           if (node.getNodeType() == ELEMENT_TYPE
           && treeElement( node.getNodeName() )
           && elementNodeIndex++ == searchIndex) {
              break;
           }
         }
       }
       return new AdapterNode(node);
     }

     public int childCount() {
       if (!compress) {
         // Indent this
         return domNode.getChildNodes().getLength();
       }
       int count = 0;
       for (int i=0; i<domNode.getChildNodes().getLength(); i++) {
          org.w3c.dom.Node node = domNode.getChildNodes().item(i);
          if (node.getNodeType() == ELEMENT_TYPE
          && treeElement( node.getNodeName() ))
          {
            // Note:
            //   Have to check for proper type.
            //   The DOCTYPE element also has the right name
            ++count;
          }
       }
       return count;
     }
   }
```

```
    // This adapter converts the current Document (a DOM) into
    // a JTree model.
    public class DomToTreeModelAdapter
      implements javax.swing.tree.TreeModel
    {
      // Basic TreeModel operations
      public Object  getRoot() {
        //System.err.println("Returning root: " +document);
        return new AdapterNode(document);
      }
      public boolean isLeaf(Object aNode) {
        // Determines whether the icon shows up to the left.
        // Return true for any node with no children
        AdapterNode node = (AdapterNode) aNode;
        if (node.childCount() > 0) return false;
        return true;
      }
      public int     getChildCount(Object parent) {
        AdapterNode node = (AdapterNode) parent;
        return node.childCount();
      }
      public Object getChild(Object parent, int index) {
        AdapterNode node = (AdapterNode) parent;
        return node.child(index);
      }
      public int getIndexOfChild(Object parent, Object child) {
        AdapterNode node = (AdapterNode) parent;
        return node.index((AdapterNode) child);
      }
      public void valueForPathChanged(TreePath path, Object newValue) {
        // Null. We won't be making changes in the GUI
        // If we did, we would ensure the new value was really new,
        // adjust the model, and then fire a TreeNodesChanged event.
      }

      /*
       * Use these methods to add and remove event listeners.
       * (Needed to satisfy TreeModel interface, but not used.)
       */
      private Vector listenerList = new Vector();
      public void addTreeModelListener(TreeModelListener listener) {
        if ( listener != null
        && ! listenerList.contains( listener ) ) {
           listenerList.addElement( listener );
        }
      }
      public void removeTreeModelListener(TreeModelListener listener) {
        if ( listener != null ) {
           listenerList.removeElement( listener );
        }
      }

      // Note: Since XML works with 1.1, this example uses Vector.
      // If coding for 1.2 or later, though, I'd use this instead:
      //   private List listenerList = new LinkedList();
      // The operations on the List are then add(), remove() and
      // iteration, via:
      //  Iterator it = listenerList.iterator();
      //  while ( it.hasNext() ) {
      //    TreeModelListener listener = (TreeModelListener) it.next();
      //    ...
      //  }

      /*
       * Invoke these methods to inform listeners of changes.
       * (Not needed for this example.)
       * Methods taken from TreeModelSupport class described at
       *   http://java.sun.com/products/jfc/tsc/articles/jtree/index.html
       * That architecture (produced by Tom Santos and Steve Wilson)
       * is more elegant. I just hacked 'em in here so they are
       * immediately at hand.
       */
      public void fireTreeNodesChanged( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
          TreeModelListener listener =
            (TreeModelListener) listeners.nextElement();
          listener.treeNodesChanged( e );
        }
      }
      public void fireTreeNodesInserted( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
           TreeModelListener listener =
             (TreeModelListener) listeners.nextElement();
           listener.treeNodesInserted( e );
        }
      }
      public void fireTreeNodesRemoved( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
          TreeModelListener listener =
            (TreeModelListener) listeners.nextElement();
          listener.treeNodesRemoved( e );
        }
      }
      public void fireTreeStructureChanged( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
          TreeModelListener listener =
            (TreeModelListener) listeners.nextElement();
          listener.treeStructureChanged( e );
        }
      }
    }
}
```

```
/*
 * @(#)DomEcho04.java   1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import java.io.File;
import java.io.IOException;

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

// Basic GUI components
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;

// GUI components for right-hand side
import javax.swing.JSplitPane;
import javax.swing.JEditorPane;

// GUI support classes
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.WindowEvent;
import java.awt.event.WindowAdapter;

// For creating borders
```

```
import javax.swing.border.EmptyBorder;
import javax.swing.border.BevelBorder;
import javax.swing.border.CompoundBorder;

// For creating a TreeModel
import javax.swing.tree.*;
import javax.swing.event.*;
import java.util.*;

public class DomEcho04  extends JPanel
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

    boolean compress = true;
    static final int windowHeight = 460;
    static final int leftWidth = 300;
    static final int rightWidth = 340;
    static final int windowWidth = leftWidth + rightWidth;

    public DomEcho04()
    {
       // Make a nice border
       EmptyBorder eb = new EmptyBorder(5,5,5,5);
       BevelBorder bb = new BevelBorder(BevelBorder.LOWERED);
       CompoundBorder cb = new CompoundBorder(eb,bb);
       this.setBorder(new CompoundBorder(cb,eb));

       // Set up the tree
       JTree tree = new JTree(new DomToTreeModelAdapter());

       // Iterate over the tree and make nodes visible
       // (Otherwise, the tree shows up fully collapsed)
       //TreePath nodePath = ???;
       //  tree.expandPath(nodePath);

       // Build left-side view
       JScrollPane treeView = new JScrollPane(tree);
       treeView.setPreferredSize(
           new Dimension( leftWidth, windowHeight ));

       // Build right-side view
       // (must be final to be referenced in inner class)
       final
       JEditorPane htmlPane = new JEditorPane("text/html","");
       htmlPane.setEditable(false);
       JScrollPane htmlView = new JScrollPane(htmlPane);
       htmlView.setPreferredSize(
           new Dimension( rightWidth, windowHeight ));

       // Wire the two views together. Use a selection listener
       // created with an anonymous inner-class adapter.
       tree.addTreeSelectionListener(
         new TreeSelectionListener() {
           public void valueChanged(TreeSelectionEvent e) {
             TreePath p = e.getNewLeadSelectionPath();
             if (p != null) {
               AdapterNode adpNode =
                  (AdapterNode) p.getLastPathComponent();
               htmlPane.setText(adpNode.content());
             }
           }
```

```
        }
      );

      // Build split-pane view
      JSplitPane splitPane =
         new JSplitPane( JSplitPane.HORIZONTAL_SPLIT,
                         treeView,
                         htmlView );
      splitPane.setContinuousLayout( true );
      splitPane.setDividerLocation( leftWidth );
      splitPane.setPreferredSize(
           new Dimension( windowWidth + 10, windowHeight+10 ));

      // Add GUI components
      this.setLayout(new BorderLayout());
      this.add("Center", splitPane );
   } // constructor

   public static void main(String argv[])
   {
       if (argv.length != 1) {
           System.err.println("Usage: java DomEcho filename");
           System.exit(1);
       }

       DocumentBuilderFactory factory =
           DocumentBuilderFactory.newInstance();
       //factory.setValidating(true);
       //factory.setNamespaceAware(true);
       try {
          DocumentBuilder builder = factory.newDocumentBuilder();
          document = builder.parse( new File(argv[0]) );
           makeFrame();

       } catch (SAXException sxe) {
          // Error generated during parsing)
          Exception  x = sxe;
          if (sxe.getException() != null)
              x = sxe.getException();
          x.printStackTrace();

       } catch (ParserConfigurationException pce) {
           // Parser with specified options can't be built
           pce.printStackTrace();

       } catch (IOException ioe) {
          // I/O error
          ioe.printStackTrace();
       }
   } // main

   public static void makeFrame() {
       // Set up a GUI framework
       JFrame frame = new JFrame("DOM Echo");
       frame.addWindowListener(
         new WindowAdapter() {
           public void windowClosing(WindowEvent e) {System.exit(0);}
         }
       );

       // Set up the tree, the views, and display it all
       final DomEcho04 echoPanel =
          new DomEcho04();
       frame.getContentPane().add("Center", echoPanel );
       frame.pack();
       Dimension screenSize =
          Toolkit.getDefaultToolkit().getScreenSize();
       int w = windowWidth + 10;
       int h = windowHeight + 10;
       frame.setLocation(screenSize.width/3 - w/2,
                         screenSize.height/2 - h/2);
       frame.setSize(w, h);
       frame.setVisible(true);
   } // makeFrame


   // An array of names for DOM node-types
   // (Array indexes = nodeType() values.)
   static final String[] typeName = {
       "none",
       "Element",
       "Attr",
       "Text",
       "CDATA",
       "EntityRef",
       "Entity",
       "ProcInstr",
       "Comment",
       "Document",
       "DocType",
       "DocFragment",
       "Notation",
   };
   static final int ELEMENT_TYPE =   1;
   static final int ATTR_TYPE =      2;
   static final int TEXT_TYPE =      3;
   static final int CDATA_TYPE =     4;
   static final int ENTITYREF_TYPE = 5;
   static final int ENTITY_TYPE =    6;
   static final int PROCINSTR_TYPE = 7;
   static final int COMMENT_TYPE =   8;
   static final int DOCUMENT_TYPE =  9;
   static final int DOCTYPE_TYPE =  10;
   static final int DOCFRAG_TYPE =  11;
   static final int NOTATION_TYPE = 12;
   // The list of elements to display in the tree
   // Could set this with a command-line argument, but
   // not much point -- the list of tree elements still
   // has to be defined internally.
   // Extra credit: Read the list from a file
   // Super-extra credit: Process a DTD and build the list.
  static String[] treeElementNames = {
       "slideshow",
       "slide",
       "title",         // For slideshow #1
       "slide-title",   // For slideshow #10
       "item",
   };
   boolean treeElement(String elementName) {
     for (int i=0; i<treeElementNames.length; i++) {
       if ( elementName.equals(treeElementNames[i]) ) return true;
     }
     return false;
```

```
  }

  // This class wraps a DOM node and returns the text we want to
  // display in the tree. It also returns children, index values,
  // and child counts.
  public class AdapterNode
  {
    org.w3c.dom.Node domNode;

    // Construct an Adapter node from a DOM node
    public AdapterNode(org.w3c.dom.Node node) {
      domNode = node;
    }

    // Return a string that identifies this node in the tree
    // *** Refer to table at top of org.w3c.dom.Node ***
    public String toString() {
      String s = typeName[domNode.getNodeType()];
      String nodeName = domNode.getNodeName();
      if (! nodeName.startsWith("#")) {
         s += ": " + nodeName;
      }
      if (compress) {
         String t = content().trim();
         int x = t.indexOf("\n");
         if (x >= 0) t = t.substring(0, x);
         s += " " + t;
         return s;
      }
      if (domNode.getNodeValue() != null) {
         if (s.startsWith("ProcInstr"))
            s += ", ";
         else
            s += ": ";
         // Trim the value to get rid of NL's at the front
         String t = domNode.getNodeValue().trim();
         int x = t.indexOf("\n");
         if (x >= 0) t = t.substring(0, x);
         s += t;
      }
      return s;
    }

    public String content() {
      String s = "";
      org.w3c.dom.NodeList nodeList = domNode.getChildNodes();
      for (int i=0; i<nodeList.getLength(); i++) {
        org.w3c.dom.Node node = nodeList.item(i);
        int type = node.getNodeType();
        AdapterNode adpNode = new AdapterNode(node); //inefficient, but works
        if (type == ELEMENT_TYPE) {
          // Skip subelements that are displayed in the tree.
          if ( treeElement(node.getNodeName()) ) continue;

          // EXTRA-CREDIT HOMEWORK:
          //   Special case the SLIDE element to use the TITLE text
          //   and ignore TITLE element when constructing the tree.

          // EXTRA-CREDIT
          //   Convert ITEM elements to html lists using
          //   <ul>, <li>, </ul> tags
          s += "<" + node.getNodeName() + ">";
          s += adpNode.content();
          s += "</" + node.getNodeName() + ">";
        } else if (type == TEXT_TYPE) {
          s += node.getNodeValue();
        } else if (type == ENTITYREF_TYPE) {
          // The content is in the TEXT node under it
          s += adpNode.content();
        } else if (type == CDATA_TYPE) {
          // The "value" has the text, same as a text node.
          //   while EntityRef has it in a text node underneath.
          //   (because EntityRef can contain multiple subelements)
          // Convert angle brackets and ampersands for display
          StringBuffer sb = new StringBuffer( node.getNodeValue() );
          for (int j=0; j<sb.length(); j++) {
            if (sb.charAt(j) == '<') {
              sb.setCharAt(j, '&');
              sb.insert(j+1, "lt;");
              j += 3;
            } else if (sb.charAt(j) == '&') {
              sb.setCharAt(j, '&');
              sb.insert(j+1, "amp;");
              j += 4;
            }
          }
          s += "<pre>" + sb + "\n</pre>";
        }
         // Ignoring these:
         //   ATTR_TYPE      -- not in the DOM tree
         //   ENTITY_TYPE    -- does not appear in the DOM
         //   PROCINSTR_TYPE -- not "data"
         //   COMMENT_TYPE   -- not "data"
         //   DOCUMENT_TYPE  -- Root node only. No data to display.
         //   DOCTYPE_TYPE   -- Appears under the root only
         //   DOCFRAG_TYPE   -- equiv. to "document" for fragments
         //   NOTATION_TYPE  -- nothing but binary data in here
      }
      return s;
    }

    /*
     * Return children, index, and count values
     */
    public int index(AdapterNode child) {
      //System.err.println("Looking for index of " + child);
      int count = childCount();
      for (int i=0; i<count; i++) {
        AdapterNode n = this.child(i);
        if (child.domNode == n.domNode) return i;
      }
      return -1; // Should never get here.
    }

    public AdapterNode child(int searchIndex) {
      //Note: JTree index is zero-based.
      org.w3c.dom.Node node =
           domNode.getChildNodes().item(searchIndex);
      if (compress) {
        // Return Nth displayable node
        int elementNodeIndex = 0;
        for (int i=0; i<domNode.getChildNodes().getLength(); i++) {
          node = domNode.getChildNodes().item(i);
          if (node.getNodeType() == ELEMENT_TYPE
          && treeElement( node.getNodeName() )
          && elementNodeIndex++ == searchIndex) {
             break;
          }
        }
      }
      return new AdapterNode(node);
    }

    public int childCount() {
      if (!compress) {
        // Indent this
        return domNode.getChildNodes().getLength();
      }
      int count = 0;
      for (int i=0; i<domNode.getChildNodes().getLength(); i++) {
         org.w3c.dom.Node node = domNode.getChildNodes().item(i);
         if (node.getNodeType() == ELEMENT_TYPE
         && treeElement( node.getNodeName() ))
         {
           // Note:
           //   Have to check for proper type.
           //   The DOCTYPE element also has the right name
           ++count;
         }
      }
      return count;
    }
  }

  // This adapter converts the current Document (a DOM) into
  // a JTree model.
  public class DomToTreeModelAdapter
    implements javax.swing.tree.TreeModel
  {
    // Basic TreeModel operations
    public Object  getRoot() {
      //System.err.println("Returning root: " +document);
      return new AdapterNode(document);
    }
    public boolean isLeaf(Object aNode) {
      // Determines whether the icon shows up to the left.
      // Return true for any node with no children
      AdapterNode node = (AdapterNode) aNode;
      if (node.childCount() > 0) return false;
      return true;
    }
    public int     getChildCount(Object parent) {
      AdapterNode node = (AdapterNode) parent;
      return node.childCount();
    }
    public Object getChild(Object parent, int index) {
      AdapterNode node = (AdapterNode) parent;
      return node.child(index);
    }
    public int getIndexOfChild(Object parent, Object child) {
      AdapterNode node = (AdapterNode) parent;
      return node.index((AdapterNode) child);
    }
    public void valueForPathChanged(TreePath path, Object newValue) {
      // Null. We won't be making changes in the GUI
      // If we did, we would ensure the new value was really new,
      // adjust the model, and then fire a TreeNodesChanged event.
    }

    /*
     * Use these methods to add and remove event listeners.
     * (Needed to satisfy TreeModel interface, but not used.)
     */
    private Vector listenerList = new Vector();
    public void addTreeModelListener(TreeModelListener listener) {
      if ( listener != null
      && ! listenerList.contains( listener ) ) {
         listenerList.addElement( listener );
      }
    }
    public void removeTreeModelListener(TreeModelListener listener) {
      if ( listener != null ) {
         listenerList.removeElement( listener );
      }
    }

    // Note: Since XML works with 1.1, this example uses Vector.
    // If coding for 1.2 or later, though, I'd use this instead:
    //   private List listenerList = new LinkedList();
    // The operations on the List are then add(), remove() and
    // iteration, via:
    //  Iterator it = listenerList.iterator();
    //  while ( it.hasNext() ) {
    //    TreeModelListener listener = (TreeModelListener) it.next();
    //    ...
    //  }

    /*
     * Invoke these methods to inform listeners of changes.
     * (Not needed for this example.)
     * Methods taken from TreeModelSupport class described at
     *   http://java.sun.com/products/jfc/tsc/articles/jtree/index.html
     * That architecture (produced by Tom Santos and Steve Wilson)
     * is more elegant. I just hacked 'em in here so they are
     * immediately at hand.
     */
    public void fireTreeNodesChanged( TreeModelEvent e ) {
      Enumeration listeners = listenerList.elements();
      while ( listeners.hasMoreElements() ) {
        TreeModelListener listener =
          (TreeModelListener) listeners.nextElement();
        listener.treeNodesChanged( e );
      }
    }
    public void fireTreeNodesInserted( TreeModelEvent e ) {
      Enumeration listeners = listenerList.elements();
      while ( listeners.hasMoreElements() ) {
         TreeModelListener listener =
           (TreeModelListener) listeners.nextElement();
         listener.treeNodesInserted( e );
      }
    }
    public void fireTreeNodesRemoved( TreeModelEvent e ) {
      Enumeration listeners = listenerList.elements();
```

```
        while ( listeners.hasMoreElements() ) {
          TreeModelListener listener =
            (TreeModelListener) listeners.nextElement();
          listener.treeNodesRemoved( e );
        }
      }
      public void fireTreeStructureChanged( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
          TreeModelListener listener =
            (TreeModelListener) listeners.nextElement();
          listener.treeStructureChanged( e );
        }
      }
    }
}
```

```
/*
 * @(#)DomEcho05.java   1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import java.io.File;
import java.io.IOException;

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
import org.w3c.dom.Element;

// Basic GUI components
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;

// GUI components for right-hand side
import javax.swing.JSplitPane;
import javax.swing.JEditorPane;

// GUI support classes
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.WindowEvent;
import java.awt.event.WindowAdapter;
```

```
// For creating borders
import javax.swing.border.EmptyBorder;
import javax.swing.border.BevelBorder;
import javax.swing.border.CompoundBorder;

// For creating a TreeModel
import javax.swing.tree.*;
import javax.swing.event.*;
import java.util.*;

public class DomEcho05  extends JPanel
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

    boolean compress = false;
    static final int windowHeight = 460;
    static final int leftWidth = 300;
    static final int rightWidth = 340;
    static final int windowWidth = leftWidth + rightWidth;

    public DomEcho05()
    {
       // Make a nice border
       EmptyBorder eb = new EmptyBorder(5,5,5,5);
       BevelBorder bb = new BevelBorder(BevelBorder.LOWERED);
       CompoundBorder cb = new CompoundBorder(eb,bb);
       this.setBorder(new CompoundBorder(cb,eb));

       // Set up the tree
       JTree tree = new JTree(new DomToTreeModelAdapter());

       // Iterate over the tree and make nodes visible
       // (Otherwise, the tree shows up fully collapsed)
       //TreePath nodePath = ???;
       //  tree.expandPath(nodePath);

       // Build left-side view
       JScrollPane treeView = new JScrollPane(tree);
       treeView.setPreferredSize(
           new Dimension( leftWidth, windowHeight ));

       // Build right-side view
       // (must be final to be referenced in inner class)
       final
       JEditorPane htmlPane = new JEditorPane("text/html","");
       htmlPane.setEditable(false);
       JScrollPane htmlView = new JScrollPane(htmlPane);
       htmlView.setPreferredSize(
           new Dimension( rightWidth, windowHeight ));

       // Wire the two views together. Use a selection listener
       // created with an anonymous inner-class adapter.
       tree.addTreeSelectionListener(
         new TreeSelectionListener() {
           public void valueChanged(TreeSelectionEvent e) {
             TreePath p = e.getNewLeadSelectionPath();
             if (p != null) {
               AdapterNode adpNode =
                  (AdapterNode) p.getLastPathComponent();
               htmlPane.setText(adpNode.content());
             }
```

```
          }
        }
      );

      // Build split-pane view
      JSplitPane splitPane =
         new JSplitPane( JSplitPane.HORIZONTAL_SPLIT,
                         treeView,
                         htmlView );
      splitPane.setContinuousLayout( true );
      splitPane.setDividerLocation( leftWidth );
      splitPane.setPreferredSize(
           new Dimension( windowWidth + 10, windowHeight+10 ));

      // Add GUI components
      this.setLayout(new BorderLayout());
      this.add("Center", splitPane );
   } // constructor

   public static void main(String argv[])
   {
       if (argv.length != 1) {
               buildDom();
               makeFrame();
               return;
               }

       DocumentBuilderFactory factory =
           DocumentBuilderFactory.newInstance();
       //factory.setValidating(true);
       //factory.setNamespaceAware(true);
       try {
          DocumentBuilder builder = factory.newDocumentBuilder();
          document = builder.parse( new File(argv[0]) );
           makeFrame();

       } catch (SAXException sxe) {
          // Error generated during parsing)
          Exception  x = sxe;
          if (sxe.getException() != null)
              x = sxe.getException();
          x.printStackTrace();

       } catch (ParserConfigurationException pce) {
           // Parser with specified options can't be built
           pce.printStackTrace();

       } catch (IOException ioe) {
          // I/O error
          ioe.printStackTrace();
       }
   } // main

   public static void makeFrame() {
       // Set up a GUI framework
       JFrame frame = new JFrame("DOM Echo");
       frame.addWindowListener(
         new WindowAdapter() {
           public void windowClosing(WindowEvent e) {System.exit(0);}
         }
       );
```

```
        // Set up the tree, the views, and display it all
        final DomEcho05 echoPanel =
           new DomEcho05();
        frame.getContentPane().add("Center", echoPanel );
        frame.pack();
        Dimension screenSize =
           Toolkit.getDefaultToolkit().getScreenSize();
        int w = windowWidth + 10;
        int h = windowHeight + 10;
        frame.setLocation(screenSize.width/3 - w/2,
                          screenSize.height/2 - h/2);
        frame.setSize(w, h);
        frame.setVisible(true);
    } // makeFrame

    public static void buildDom()
    {
        DocumentBuilderFactory factory =
           DocumentBuilderFactory.newInstance();
        try {
          DocumentBuilder builder = factory.newDocumentBuilder();
          document = builder.newDocument();  // Create from whole cloth

          Element root =
                  (Element) document.createElement("rootElement");
          document.appendChild(root);
          root.appendChild( document.createTextNode("Some") );
          root.appendChild( document.createTextNode(" ")    );
          root.appendChild( document.createTextNode("text") );


        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        }
    } // buildDom

    // An array of names for DOM node-types
    // (Array indexes = nodeType() values.)
    static final String[] typeName = {
        "none",
        "Element",
        "Attr",
        "Text",
        "CDATA",
        "EntityRef",
        "Entity",
        "ProcInstr",
        "Comment",
        "Document",
        "DocType",
        "DocFragment",
        "Notation",
    };
    static final int ELEMENT_TYPE =   1;
    static final int ATTR_TYPE =      2;
    static final int TEXT_TYPE =      3;
    static final int CDATA_TYPE =     4;
    static final int ENTITYREF_TYPE = 5;
    static final int ENTITY_TYPE =    6;
    static final int PROCINSTR_TYPE = 7;
```

```
   static final int COMMENT_TYPE =   8;
   static final int DOCUMENT_TYPE =  9;
   static final int DOCTYPE_TYPE =  10;
   static final int DOCFRAG_TYPE =  11;
   static final int NOTATION_TYPE = 12;
   // The list of elements to display in the tree
   // Could set this with a command-line argument, but
   // not much point -- the list of tree elements still
   // has to be defined internally.
   // Extra credit: Read the list from a file
   // Super-extra credit: Process a DTD and build the list.
  static String[] treeElementNames = {
       "slideshow",
       "slide",
       "title",         // For slideshow #1
       "slide-title",   // For slideshow #10
       "item",
   };
   boolean treeElement(String elementName) {
     for (int i=0; i<treeElementNames.length; i++) {
       if ( elementName.equals(treeElementNames[i]) ) return true;
     }
     return false;
   }

   // This class wraps a DOM node and returns the text we want to
   // display in the tree. It also returns children, index values,
   // and child counts.
   public class AdapterNode
   {
     org.w3c.dom.Node domNode;

     // Construct an Adapter node from a DOM node
     public AdapterNode(org.w3c.dom.Node node) {
       domNode = node;
     }

     // Return a string that identifies this node in the tree
     // *** Refer to table at top of org.w3c.dom.Node ***
     public String toString() {
       String s = typeName[domNode.getNodeType()];
       String nodeName = domNode.getNodeName();
       if (! nodeName.startsWith("#")) {
          s += ": " + nodeName;
       }
       if (compress) {
          String t = content().trim();
          int x = t.indexOf("\n");
          if (x >= 0) t = t.substring(0, x);
          s += " " + t;
          return s;
       }
       if (domNode.getNodeValue() != null) {
          if (s.startsWith("ProcInstr"))
             s += ", ";
          else
             s += ": ";
          // Trim the value to get rid of NL's at the front
          String t = domNode.getNodeValue().trim();
          int x = t.indexOf("\n");
          if (x >= 0) t = t.substring(0, x);
          s += t;
       }
       return s;
     }

     public String content() {
       String s = "";
       org.w3c.dom.NodeList nodeList = domNode.getChildNodes();
       for (int i=0; i<nodeList.getLength(); i++) {
         org.w3c.dom.Node node = nodeList.item(i);
         int type = node.getNodeType();
         AdapterNode adpNode = new AdapterNode(node); //inefficient, but works
         if (type == ELEMENT_TYPE) {
           // Skip subelements that are displayed in the tree.
           if ( treeElement(node.getNodeName()) ) continue;

           // EXTRA-CREDIT HOMEWORK:
           //   Special case the SLIDE element to use the TITLE text
           //   and ignore TITLE element when constructing the tree.

           // EXTRA-CREDIT
           //   Convert ITEM elements to html lists using
           //   <ul>, <li>, </ul> tags

           s += "<" + node.getNodeName() + ">";
           s += adpNode.content();
           s += "</" + node.getNodeName() + ">";
         } else if (type == TEXT_TYPE) {
           s += node.getNodeValue();
         } else if (type == ENTITYREF_TYPE) {
           // The content is in the TEXT node under it
           s += adpNode.content();
         } else if (type == CDATA_TYPE) {
           // The "value" has the text, same as a text node.
           //   while EntityRef has it in a text node underneath.
           //   (because EntityRef can contain multiple subelements)
           // Convert angle brackets and ampersands for display
           StringBuffer sb = new StringBuffer( node.getNodeValue() );
           for (int j=0; j<sb.length(); j++) {
             if (sb.charAt(j) == '<') {
               sb.setCharAt(j, '&');
               sb.insert(j+1, "lt;");
               j += 3;
             } else if (sb.charAt(j) == '&') {
               sb.setCharAt(j, '&');
               sb.insert(j+1, "amp;");
               j += 4;
             }
           }
           s += "<pre>" + sb + "\n</pre>";
         }
          // Ignoring these:
          //   ATTR_TYPE      -- not in the DOM tree
          //   ENTITY_TYPE    -- does not appear in the DOM
          //   PROCINSTR_TYPE -- not "data"
          //   COMMENT_TYPE   -- not "data"
          //   DOCUMENT_TYPE  -- Root node only. No data to display.
          //   DOCTYPE_TYPE   -- Appears under the root only
          //   DOCFRAG_TYPE   -- equiv. to "document" for fragments
          //   NOTATION_TYPE  -- nothing but binary data in here
       }
       return s;
     }

     /*
      * Return children, index, and count values
      */
     public int index(AdapterNode child) {
       //System.err.println("Looking for index of " + child);
       int count = childCount();
       for (int i=0; i<count; i++) {
         AdapterNode n = this.child(i);
         if (child.domNode == n.domNode) return i;
       }
       return -1; // Should never get here.
     }

     public AdapterNode child(int searchIndex) {
       //Note: JTree index is zero-based.
       org.w3c.dom.Node node =
            domNode.getChildNodes().item(searchIndex);
       if (compress) {
         // Return Nth displayable node
         int elementNodeIndex = 0;
         for (int i=0; i<domNode.getChildNodes().getLength(); i++) {
           node = domNode.getChildNodes().item(i);
           if (node.getNodeType() == ELEMENT_TYPE
           && treeElement( node.getNodeName() )
           && elementNodeIndex++ == searchIndex) {
              break;
           }
         }
       }
       return new AdapterNode(node);
     }

     public int childCount() {
       if (!compress) {
         // Indent this
         return domNode.getChildNodes().getLength();
       }
       int count = 0;
       for (int i=0; i<domNode.getChildNodes().getLength(); i++) {
          org.w3c.dom.Node node = domNode.getChildNodes().item(i);
          if (node.getNodeType() == ELEMENT_TYPE
          && treeElement( node.getNodeName() ))
          {
            // Note:
            //   Have to check for proper type.
            //   The DOCTYPE element also has the right name
            ++count;
          }
       }
       return count;
     }
   }

   // This adapter converts the current Document (a DOM) into
   // a JTree model.
   public class DomToTreeModelAdapter
     implements javax.swing.tree.TreeModel
   {
     // Basic TreeModel operations
     public Object  getRoot() {
       //System.err.println("Returning root: " +document);
       return new AdapterNode(document);
     }
     public boolean isLeaf(Object aNode) {
       // Determines whether the icon shows up to the left.
       // Return true for any node with no children
       AdapterNode node = (AdapterNode) aNode;
       if (node.childCount() > 0) return false;
       return true;
     }
     public int     getChildCount(Object parent) {
       AdapterNode node = (AdapterNode) parent;
       return node.childCount();
     }
     public Object getChild(Object parent, int index) {
       AdapterNode node = (AdapterNode) parent;
       return node.child(index);
     }
     public int getIndexOfChild(Object parent, Object child) {
       AdapterNode node = (AdapterNode) parent;
       return node.index((AdapterNode) child);
     }
     public void valueForPathChanged(TreePath path, Object newValue) {
       // Null. We won't be making changes in the GUI
       // If we did, we would ensure the new value was really new,
       // adjust the model, and then fire a TreeNodesChanged event.
     }

     /*
      * Use these methods to add and remove event listeners.
      * (Needed to satisfy TreeModel interface, but not used.)
      */
     private Vector listenerList = new Vector();
     public void addTreeModelListener(TreeModelListener listener) {
       if ( listener != null
       && ! listenerList.contains( listener ) ) {
          listenerList.addElement( listener );
       }
     }
     public void removeTreeModelListener(TreeModelListener listener) {
       if ( listener != null ) {
          listenerList.removeElement( listener );
       }
     }

     // Note: Since XML works with 1.1, this example uses Vector.
     // If coding for 1.2 or later, though, I'd use this instead:
     //   private List listenerList = new LinkedList();
     // The operations on the List are then add(), remove() and
     // iteration, via:
     //  Iterator it = listenerList.iterator();
     //  while ( it.hasNext() ) {
     //    TreeModelListener listener = (TreeModelListener) it.next();
     //    ...
     //  }

     /*
      * Invoke these methods to inform listeners of changes.
      * (Not needed for this example.)
```

```
        * Methods taken from TreeModelSupport class described at
        *   http://java.sun.com/products/jfc/tsc/articles/jtree/index.html
        * That architecture (produced by Tom Santos and Steve Wilson)
        * is more elegant. I just hacked 'em in here so they are
        * immediately at hand.
        */
       public void fireTreeNodesChanged( TreeModelEvent e ) {
         Enumeration listeners = listenerList.elements();
         while ( listeners.hasMoreElements() ) {
           TreeModelListener listener =
             (TreeModelListener) listeners.nextElement();
           listener.treeNodesChanged( e );
         }
       }
       public void fireTreeNodesInserted( TreeModelEvent e ) {
         Enumeration listeners = listenerList.elements();
         while ( listeners.hasMoreElements() ) {
            TreeModelListener listener =
              (TreeModelListener) listeners.nextElement();
            listener.treeNodesInserted( e );
         }
       }
       public void fireTreeNodesRemoved( TreeModelEvent e ) {
         Enumeration listeners = listenerList.elements();
         while ( listeners.hasMoreElements() ) {
           TreeModelListener listener =
             (TreeModelListener) listeners.nextElement();
           listener.treeNodesRemoved( e );
         }
       }
       public void fireTreeStructureChanged( TreeModelEvent e ) {
         Enumeration listeners = listenerList.elements();
         while ( listeners.hasMoreElements() ) {
           TreeModelListener listener =
             (TreeModelListener) listeners.nextElement();
           listener.treeStructureChanged( e );
         }
       }
     }
 }
```

```
/*
 * @(#)DomEcho06.java   1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import java.io.File;
import java.io.IOException;

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
import org.w3c.dom.Element;

// Basic GUI components
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;

// GUI components for right-hand side
import javax.swing.JSplitPane;
import javax.swing.JEditorPane;

// GUI support classes
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.WindowEvent;
import java.awt.event.WindowAdapter;
```

```
// For creating borders
import javax.swing.border.EmptyBorder;
import javax.swing.border.BevelBorder;
import javax.swing.border.CompoundBorder;

// For creating a TreeModel
import javax.swing.tree.*;
import javax.swing.event.*;
import java.util.*;

public class DomEcho06  extends JPanel
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

    boolean compress = false;
    static final int windowHeight = 460;
    static final int leftWidth = 300;
    static final int rightWidth = 340;
    static final int windowWidth = leftWidth + rightWidth;

    public DomEcho06()
    {
       // Make a nice border
       EmptyBorder eb = new EmptyBorder(5,5,5,5);
       BevelBorder bb = new BevelBorder(BevelBorder.LOWERED);
       CompoundBorder cb = new CompoundBorder(eb,bb);
       this.setBorder(new CompoundBorder(cb,eb));

       // Set up the tree
       JTree tree = new JTree(new DomToTreeModelAdapter());

       // Iterate over the tree and make nodes visible
       // (Otherwise, the tree shows up fully collapsed)
       //TreePath nodePath = ???;
       //  tree.expandPath(nodePath);

       // Build left-side view
       JScrollPane treeView = new JScrollPane(tree);
       treeView.setPreferredSize(
           new Dimension( leftWidth, windowHeight ));

       // Build right-side view
       // (must be final to be referenced in inner class)
       final
       JEditorPane htmlPane = new JEditorPane("text/html","");
       htmlPane.setEditable(false);
       JScrollPane htmlView = new JScrollPane(htmlPane);
       htmlView.setPreferredSize(
           new Dimension( rightWidth, windowHeight ));

       // Wire the two views together. Use a selection listener
       // created with an anonymous inner-class adapter.
       tree.addTreeSelectionListener(
         new TreeSelectionListener() {
           public void valueChanged(TreeSelectionEvent e) {
             TreePath p = e.getNewLeadSelectionPath();
             if (p != null) {
               AdapterNode adpNode =
                  (AdapterNode) p.getLastPathComponent();
               htmlPane.setText(adpNode.content());
             }
```

```
          }
        }
      );

      // Build split-pane view
      JSplitPane splitPane =
         new JSplitPane( JSplitPane.HORIZONTAL_SPLIT,
                         treeView,
                         htmlView );
      splitPane.setContinuousLayout( true );
      splitPane.setDividerLocation( leftWidth );
      splitPane.setPreferredSize(
           new Dimension( windowWidth + 10, windowHeight+10 ));

      // Add GUI components
      this.setLayout(new BorderLayout());
      this.add("Center", splitPane );
   } // constructor

   public static void main(String argv[])
   {
       if (argv.length != 1) {
               buildDom();
               makeFrame();
               return;
               }

       DocumentBuilderFactory factory =
           DocumentBuilderFactory.newInstance();
       //factory.setValidating(true);
       //factory.setNamespaceAware(true);
       try {
          DocumentBuilder builder = factory.newDocumentBuilder();
          document = builder.parse( new File(argv[0]) );
           makeFrame();

       } catch (SAXException sxe) {
          // Error generated during parsing)
          Exception  x = sxe;
          if (sxe.getException() != null)
              x = sxe.getException();
          x.printStackTrace();

       } catch (ParserConfigurationException pce) {
           // Parser with specified options can't be built
           pce.printStackTrace();

       } catch (IOException ioe) {
          // I/O error
          ioe.printStackTrace();
       }
   } // main

   public static void makeFrame() {
       // Set up a GUI framework
       JFrame frame = new JFrame("DOM Echo");
       frame.addWindowListener(
         new WindowAdapter() {
           public void windowClosing(WindowEvent e) {System.exit(0);}
         }
       );
```

```
        // Set up the tree, the views, and display it all
        final DomEcho06 echoPanel =
           new DomEcho06();
        frame.getContentPane().add("Center", echoPanel );
        frame.pack();
        Dimension screenSize =
           Toolkit.getDefaultToolkit().getScreenSize();
        int w = windowWidth + 10;
        int h = windowHeight + 10;
        frame.setLocation(screenSize.width/3 - w/2,
                          screenSize.height/2 - h/2);
        frame.setSize(w, h);
        frame.setVisible(true);
    } // makeFrame

    public static void buildDom()
    {
        DocumentBuilderFactory factory =
           DocumentBuilderFactory.newInstance();
        try {
          DocumentBuilder builder = factory.newDocumentBuilder();
          document = builder.newDocument();  // Create from whole cloth

          Element root =
                  (Element) document.createElement("rootElement");
          document.appendChild(root);
          root.appendChild( document.createTextNode("Some") );
          root.appendChild( document.createTextNode(" ")    );
          root.appendChild( document.createTextNode("text") );

          // normalize text representation
          // getDocumentElement() returns the document's root node
          document.getDocumentElement().normalize();


        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        }
    } // buildDom

    // An array of names for DOM node-types
    // (Array indexes = nodeType() values.)
    static final String[] typeName = {
        "none",
        "Element",
        "Attr",
        "Text",
        "CDATA",
        "EntityRef",
        "Entity",
        "ProcInstr",
        "Comment",
        "Document",
        "DocType",
        "DocFragment",
        "Notation",
    };
    static final int ELEMENT_TYPE =   1;
    static final int ATTR_TYPE =      2;
    static final int TEXT_TYPE =      3;
```

```
   static final int CDATA_TYPE =     4;
   static final int ENTITYREF_TYPE = 5;
   static final int ENTITY_TYPE =    6;
   static final int PROCINSTR_TYPE = 7;
   static final int COMMENT_TYPE =   8;
   static final int DOCUMENT_TYPE =  9;
   static final int DOCTYPE_TYPE =  10;
   static final int DOCFRAG_TYPE =  11;
   static final int NOTATION_TYPE = 12;
   // The list of elements to display in the tree
   // Could set this with a command-line argument, but
   // not much point -- the list of tree elements still
   // has to be defined internally.
   // Extra credit: Read the list from a file
   // Super-extra credit: Process a DTD and build the list.
  static String[] treeElementNames = {
       "slideshow",
       "slide",
       "title",         // For slideshow #1
       "slide-title",   // For slideshow #10
       "item",
   };
   boolean treeElement(String elementName) {
     for (int i=0; i<treeElementNames.length; i++) {
       if ( elementName.equals(treeElementNames[i]) ) return true;
     }
     return false;
   }

   // This class wraps a DOM node and returns the text we want to
   // display in the tree. It also returns children, index values,
   // and child counts.
   public class AdapterNode
   {
     org.w3c.dom.Node domNode;

     // Construct an Adapter node from a DOM node
     public AdapterNode(org.w3c.dom.Node node) {
       domNode = node;
     }

     // Return a string that identifies this node in the tree
     // *** Refer to table at top of org.w3c.dom.Node ***
     public String toString() {
       String s = typeName[domNode.getNodeType()];
       String nodeName = domNode.getNodeName();
       if (! nodeName.startsWith("#")) {
          s += ": " + nodeName;
       }
       if (compress) {
          String t = content().trim();
          int x = t.indexOf("\n");
          if (x >= 0) t = t.substring(0, x);
          s += " " + t;
          return s;
       }
       if (domNode.getNodeValue() != null) {
          if (s.startsWith("ProcInstr"))
             s += ", ";
          else
             s += ": ";
```

```
         // Trim the value to get rid of NL's at the front
         String t = domNode.getNodeValue().trim();
         int x = t.indexOf("\n");
         if (x >= 0) t = t.substring(0, x);
         s += t;
      }
      return s;
    }

    public String content() {
      String s = "";
      org.w3c.dom.NodeList nodeList = domNode.getChildNodes();
      for (int i=0; i<nodeList.getLength(); i++) {
        org.w3c.dom.Node node = nodeList.item(i);
        int type = node.getNodeType();
        AdapterNode adpNode = new AdapterNode(node); //inefficient, but works
        if (type == ELEMENT_TYPE) {
          // Skip subelements that are displayed in the tree.
          if ( treeElement(node.getNodeName()) ) continue;

          // EXTRA-CREDIT HOMEWORK:
          //   Special case the SLIDE element to use the TITLE text
          //   and ignore TITLE element when constructing the tree.

          // EXTRA-CREDIT
          //   Convert ITEM elements to html lists using
          //   <ul>, <li>, </ul> tags

          s += "<" + node.getNodeName() + ">";
          s += adpNode.content();
          s += "</" + node.getNodeName() + ">";
        } else if (type == TEXT_TYPE) {
          s += node.getNodeValue();
        } else if (type == ENTITYREF_TYPE) {
          // The content is in the TEXT node under it
          s += adpNode.content();
        } else if (type == CDATA_TYPE) {
          // The "value" has the text, same as a text node.
          //   while EntityRef has it in a text node underneath.
          //   (because EntityRef can contain multiple subelements)
          // Convert angle brackets and ampersands for display
          StringBuffer sb = new StringBuffer( node.getNodeValue() );
          for (int j=0; j<sb.length(); j++) {
            if (sb.charAt(j) == '<') {
              sb.setCharAt(j, '&');
              sb.insert(j+1, "lt;");
              j += 3;
            } else if (sb.charAt(j) == '&') {
              sb.setCharAt(j, '&');
              sb.insert(j+1, "amp;");
              j += 4;
            }
          }
          s += "<pre>" + sb + "\n</pre>";
        }
         // Ignoring these:
         //   ATTR_TYPE      -- not in the DOM tree
         //   ENTITY_TYPE    -- does not appear in the DOM
         //   PROCINSTR_TYPE -- not "data"
         //   COMMENT_TYPE   -- not "data"
         //   DOCUMENT_TYPE  -- Root node only. No data to display.
         //   DOCTYPE_TYPE   -- Appears under the root only
         //   DOCFRAG_TYPE   -- equiv. to "document" for fragments
         //   NOTATION_TYPE  -- nothing but binary data in here
      }
      return s;
    }

    /*
     * Return children, index, and count values
     */
    public int index(AdapterNode child) {
      //System.err.println("Looking for index of " + child);
      int count = childCount();
      for (int i=0; i<count; i++) {
        AdapterNode n = this.child(i);
        if (child.domNode == n.domNode) return i;
      }
      return -1; // Should never get here.
    }

    public AdapterNode child(int searchIndex) {
      //Note: JTree index is zero-based.
      org.w3c.dom.Node node =
           domNode.getChildNodes().item(searchIndex);
      if (compress) {
        // Return Nth displayable node
        int elementNodeIndex = 0;
        for (int i=0; i<domNode.getChildNodes().getLength(); i++) {
          node = domNode.getChildNodes().item(i);
          if (node.getNodeType() == ELEMENT_TYPE
          && treeElement( node.getNodeName() )
          && elementNodeIndex++ == searchIndex) {
             break;
          }
        }
      }
      return new AdapterNode(node);
    }

    public int childCount() {
      if (!compress) {
        // Indent this
        return domNode.getChildNodes().getLength();
      }
      int count = 0;
      for (int i=0; i<domNode.getChildNodes().getLength(); i++) {
         org.w3c.dom.Node node = domNode.getChildNodes().item(i);
         if (node.getNodeType() == ELEMENT_TYPE
         && treeElement( node.getNodeName() ))
         {
           // Note:
           //   Have to check for proper type.
           //   The DOCTYPE element also has the right name
           ++count;
         }
      }
      return count;
    }
  }

  // This adapter converts the current Document (a DOM) into
  // a JTree model.
  public class DomToTreeModelAdapter
    implements javax.swing.tree.TreeModel
  {
    // Basic TreeModel operations
    public Object  getRoot() {
      //System.err.println("Returning root: " +document);
      return new AdapterNode(document);
    }
    public boolean isLeaf(Object aNode) {
      // Determines whether the icon shows up to the left.
      // Return true for any node with no children
      AdapterNode node = (AdapterNode) aNode;
      if (node.childCount() > 0) return false;
      return true;
    }
    public int     getChildCount(Object parent) {
      AdapterNode node = (AdapterNode) parent;
      return node.childCount();
    }
    public Object getChild(Object parent, int index) {
      AdapterNode node = (AdapterNode) parent;
      return node.child(index);
    }
    public int getIndexOfChild(Object parent, Object child) {
      AdapterNode node = (AdapterNode) parent;
      return node.index((AdapterNode) child);
    }
    public void valueForPathChanged(TreePath path, Object newValue) {
      // Null. We won't be making changes in the GUI
      // If we did, we would ensure the new value was really new,
      // adjust the model, and then fire a TreeNodesChanged event.
    }

    /*
     * Use these methods to add and remove event listeners.
     * (Needed to satisfy TreeModel interface, but not used.)
     */
    private Vector listenerList = new Vector();
    public void addTreeModelListener(TreeModelListener listener) {
      if ( listener != null
      && ! listenerList.contains( listener ) ) {
         listenerList.addElement( listener );
      }
    }
    public void removeTreeModelListener(TreeModelListener listener) {
      if ( listener != null ) {
         listenerList.removeElement( listener );
      }
    }

    // Note: Since XML works with 1.1, this example uses Vector.
    // If coding for 1.2 or later, though, I'd use this instead:
    //   private List listenerList = new LinkedList();
    // The operations on the List are then add(), remove() and
    // iteration, via:
    //  Iterator it = listenerList.iterator();
    //  while ( it.hasNext() ) {
    //    TreeModelListener listener = (TreeModelListener) it.next();
    //    ...
    //  }
```

```
      /*
       * Invoke these methods to inform listeners of changes.
       * (Not needed for this example.)
       * Methods taken from TreeModelSupport class described at
       *   http://java.sun.com/products/jfc/tsc/articles/jtree/index.html
       * That architecture (produced by Tom Santos and Steve Wilson)
       * is more elegant. I just hacked 'em in here so they are
       * immediately at hand.
       */
      public void fireTreeNodesChanged( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
          TreeModelListener listener =
            (TreeModelListener) listeners.nextElement();
          listener.treeNodesChanged( e );
        }
      }
      public void fireTreeNodesInserted( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
           TreeModelListener listener =
             (TreeModelListener) listeners.nextElement();
           listener.treeNodesInserted( e );
        }
      }
      public void fireTreeNodesRemoved( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
          TreeModelListener listener =
            (TreeModelListener) listeners.nextElement();
          listener.treeNodesRemoved( e );
        }
      }
      public void fireTreeStructureChanged( TreeModelEvent e ) {
        Enumeration listeners = listenerList.elements();
        while ( listeners.hasMoreElements() ) {
          TreeModelListener listener =
            (TreeModelListener) listeners.nextElement();
          listener.treeStructureChanged( e );
        }
      }
    }
}
```

```
dn: cn=Fred Flinstone,mail=fred@barneys.house
modifytimestamp: 20010409210816Z
cn: Fred Flinstone
xmozillanickname: Fred
mail: fred@barneys.house
xmozillausehtmlmail: TRUE
givenname: Fred
sn: Flinstone
telephonenumber: 999-Quarry
homephone: 999-BedrockLane
facsimiletelephonenumber: 888-Squawk
pagerphone: 777-pager
cellphone: 555-cell
xmozillaanyphone: 999-Quarry
objectclass: top
objectclass: person
```

```
/*
 * @(#)AddressBookReader.java 1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;


/**
 * AddressBookReader -- an application that reads an address book file
 * exported from Netscape Messenger using the Line Delimited Interchange
 * Format (LDIF).
 * <p>
 * LDIF address book files have this format:<pre>
 *   dn: cn=FirstName LastName,mail=emailAddress
 *   modifytimestamp: 20010328014700Z
 *   cn: FirstName LastName  --display name (concatenation of givenname+sn)
 *   xmozillanickname: Fred        --------+
 *   mail: fred                            |
 *   xmozillausehtmlmail: TRUE             +-- We care about these
 *   givenname: Fred                       |
 *   sn: Flintstone   --(surname)          |
 *   telephonenumber: 999-Quarry           |
 *   homephone: 999-BedrockLane            |
 *   facsimiletelephonenumber: 888-Squawk  |
 *   pagerphone: 777-pager                 |
 *   cellphone: 666-cell           --------+
 *   xmozillaanyphone: Work#
 *   objectclass: top
 *   objectclass: person
 * </pre>
 *
 * @author Eric Armstrong
 */
public class AddressBookReader01
{
    public static void main (String argv [])
    {
        // Check the arguments
        if (argv.length != 1) {
            System.err.println ("Usage: java AddressBookReader filename");
            System.exit (1);
        }
        String filename = argv[0];
        File f = new File(filename);
        AddressBookReader01 reader = new AddressBookReader01();
        reader.parse(f);
    }


    /** Parse the input */
    public void parse(File f)
    {
        try {
            // Get an efficient reader for the file

            FileReader r = new FileReader(f);

            BufferedReader br = new BufferedReader(r);

            // Read the file and display it's contents.
            String line = br.readLine();
            while (null != (line = br.readLine())) {
              if (line.startsWith("xmozillanickname: ")) break;
            }

            output("nickname", "xmozillanickname", line);
            line = br.readLine();
            output("email",    "mail",             line);
            line = br.readLine();
            output("html",     "xmozillausehtmlmail", line);
            line = br.readLine();
            output("firstname","givenname",        line);
            line = br.readLine();
            output("lastname", "sn",               line);
            line = br.readLine();
            output("work",     "telephonenumber",  line);
            line = br.readLine();
            output("home",     "homephone",        line);
            line = br.readLine();
            output("fax",      "facsimiletelephonenumber", line);
            line = br.readLine();
            output("pager",    "pagerphone",       line);
            line = br.readLine();
            output("cell",     "cellphone",        line);

        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    void output(String name, String prefix, String line)

    {
      int startIndex = prefix.length() + 2; // 2=length of ": " after the name
      String text = line.substring(startIndex);
      System.out.println(name + ": " + text);

    }

}
```

```
Running AddressBookReader01 ../samples/PersonalAddressBook.ldif
nickname: Fred
email: fred@barneys.house
html: TRUE
firstname: Fred
lastname: Flinstone
work: 999-Quarry
home: 999-BedrockLane
fax: 888-Squawk
pager: 777-pager
cell: 555-cell
```

```
Running TransformationApp04 ../samples/PersonalAddressBook.ldif
<?xml version="1.0" encoding="UTF-8"?>
<addressbook>
    <nickname>Fred</nickname>
    <email>fred@barneys.house</email>
    <html>TRUE</html>
    <firstname>Fred</firstname>
    <lastname>Flinstone</lastname>
    <work>999-Quarry</work>
    <home>999-BedrockLane</home>
    <fax>888-Squawk</fax>
    <pager>777-pager</pager>
    <cell>555-cell</cell>
```

```
/*
 * @(#)AddressBookReader.java 1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.AttributesImpl;

/**
 * AddressBookReader -- an application that reads an address book file
 * exported from Netscape Messenger using the Line Delimited Interchange
 * Format (LDIF).
 * <p>
 * LDIF address book files have this format:<pre>
 *   dn: cn=FirstName LastName,mail=emailAddress
 *   modifytimestamp: 20010328014700Z
 *   cn: FirstName LastName  --display name (concatenation of givenname+sn)
 *   xmozillanickname: Fred        --------+
 *   mail: fred                            |
 *   xmozillausehtmlmail: TRUE             +-- We care about these
 *   givenname: Fred                       |
 *   sn: Flintstone   --(surname)          |
 *   telephonenumber: 999-Quarry           |
 *   homephone: 999-BedrockLane            |
 *   facsimiletelephonenumber: 888-Squawk  |
 *   pagerphone: 777-pager                 |
 *   cellphone: 666-cell           --------+
 *   xmozillaanyphone: Work#
 *   objectclass: top
 *   objectclass: person
 * </pre>
 *
 * @author Eric Armstrong
 */
```

```
public class AddressBookReader02
  implements XMLReader

{

    ContentHandler handler;

    // We're not doing namespaces, and we have no
    // attributes on our elements.
    String nsu = "";  // NamespaceURI
    Attributes atts = new AttributesImpl();
    String rootElement = "addressbook";

    String indent = "\n    "; // for readability!


    /** Parse the input */
    public void parse(InputSource input)
    throws IOException, SAXException

    {
        try {
            // Get an efficient reader for the file

            java.io.Reader r = input.getCharacterStream();

            BufferedReader br = new BufferedReader(r);

            // Read the file and display it's contents.
            String line = br.readLine();
            while (null != (line = br.readLine())) {
              if (line.startsWith("xmozillanickname: ")) break;
            }

            if (handler==null) {
              throw new SAXException("No content handler");
            }
            // Note:
            // We're ignoring setDocumentLocator(), as well
            handler.startDocument();
            handler.startElement(nsu, rootElement, rootElement, atts);

            output("nickname", "xmozillanickname", line);
            line = br.readLine();
            output("email",    "mail",             line);
            line = br.readLine();
            output("html",     "xmozillausehtmlmail", line);
            line = br.readLine();
            output("firstname","givenname",        line);
            line = br.readLine();
            output("lastname", "sn",               line);
            line = br.readLine();
            output("work",     "telephonenumber",  line);
            line = br.readLine();
            output("home",     "homephone",        line);
            line = br.readLine();
            output("fax",      "facsimiletelephonenumber", line);
            line = br.readLine();
            output("pager",    "pagerphone",       line);
            line = br.readLine();
            output("cell",     "cellphone",        line);

        handler.ignorableWhitespace("\n".toCharArray(),
                                0, // start index
                                1  // length
                                );
          handler.endElement(nsu, rootElement, rootElement);
          handler.endDocument();

        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    void output(String name, String prefix, String line)

    throws SAXException

    {
      int startIndex = prefix.length() + 2; // 2=length of ": " after the name
      String text = line.substring(startIndex);

      int textLength = line.length() - startIndex;
      handler.ignorableWhitespace(indent.toCharArray(),
                              0, // start index
                              indent.length()
                              );
      handler.startElement(nsu, name, name /*"qName"*/, atts);
      handler.characters(line.toCharArray(),
                         startIndex,
                         textLength);
      handler.endElement(nsu, name, name);

    }


    /** Allow an application to register a content event handler. */
    public void setContentHandler(ContentHandler handler) {
      this.handler = handler;
    }

    /** Return the current content handler. */
    public ContentHandler getContentHandler() {
      return this.handler;
    }

    //===============================================
    // IMPLEMENT THESE FOR A ROBUST APP
    //===============================================
    /** Allow an application to register an error event handler. */
    public void setErrorHandler(ErrorHandler handler)
    { }

    /** Return the current error handler. */
    public ErrorHandler getErrorHandler()
    { return null; }

    //===============================================
    // IGNORE THESE
    //===============================================
    /** Parse an XML document from a system identifier (URI). */
    public void parse(String systemId)
    throws IOException, SAXException
    { }
    /** Return the current DTD handler. */
    public DTDHandler getDTDHandler()
    { return null; }

    /** Return the current entity resolver. */
    public EntityResolver getEntityResolver()
    { return null; }

    /** Allow an application to register an entity resolver. */
    public void setEntityResolver(EntityResolver resolver)
    { }

    /** Allow an application to register a DTD event handler. */
    public void setDTDHandler(DTDHandler handler)
    { }

    /** Look up the value of a property. */
    public Object getProperty(java.lang.String name)
    { return null; }

    /** Set the value of a property. */
    public void setProperty(java.lang.String name, java.lang.Object value)
    { }

    /** Set the state of a feature. */
    public void setFeature(java.lang.String name, boolean value)
    { }

    /** Look up the value of a feature. */
    public boolean getFeature(java.lang.String name)
    { return false; }
}
```

```
/*
 * @(#)Stylizer.java 1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

// For write operation
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;

import java.io.*;

public class Stylizer
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

    public static void main (String argv [])
    {
        if (argv.length != 2) {
            System.err.println ("Usage: java Stylizer stylesheet xmlfile");
            System.exit (1);
        }
        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        //factory.setNamespaceAware(true);
        //factory.setValidating(true);

        try {
            File stylesheet = new File(argv[0]);
            File datafile   = new File(argv[1]);

            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(datafile);

            // Use a Transformer for output
            TransformerFactory tFactory =
                TransformerFactory.newInstance();
            StreamSource stylesource = new StreamSource(stylesheet);
            Transformer transformer = tFactory.newTransformer(stylesource);

            DOMSource source = new DOMSource(document);
            StreamResult result = new StreamResult(System.out);
            transformer.transform(source, result);

        } catch (TransformerConfigurationException tce) {
           // Error generated by the parser
           System.out.println ("\n** Transformer Factory error");
           System.out.println("   " + tce.getMessage() );

           // Use the contained exception, if any
           Throwable x = tce;
           if (tce.getException() != null)
               x = tce.getException();
           x.printStackTrace();

        } catch (TransformerException te) {
           // Error generated by the parser
           System.out.println ("\n** Transformation error");
           System.out.println("   " + te.getMessage() );

           // Use the contained exception, if any
           Throwable x = te;
           if (te.getException() != null)
               x = te.getException();
           x.printStackTrace();

         } catch (SAXException sxe) {
           // Error generated by this application
           // (or a parser-initialization error)
           Exception  x = sxe;
           if (sxe.getException() != null)
               x = sxe.getException();
           x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
           // I/O error
           ioe.printStackTrace();
        }

    } // main
```

```
}
```

The First Major Section This section will introduce a subsection. The Subsection Heading This is the text of the subsection.

```xml
<?xml version="1.0"?>
<ARTICLE>
  <TITLE>A Sample Article</TITLE>
  <SECT>The First Major Section
     <PARA>This section will introduce a subsection.</PARA>
     <SECT>The Subsection Heading
       <PARA>This is the text of the subsection.
       </PARA>
     </SECT>
  </SECT>
</ARTICLE>
```

Error: Sections can only be nested 2 deep.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >
  <xsl:output method="html"/>

  <xsl:template match="/">
    <html><body>
       <xsl:apply-templates/>
    </body></html>
  </xsl:template>

  <xsl:template match="/ARTICLE/TITLE">
    <h1 align="center"> <xsl:apply-templates/> </h1>
  </xsl:template>

  <!-- Top Level Heading -->
  <xsl:template match="/ARTICLE/SECT">
      <h1> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </h1>
      <xsl:apply-templates select="SECT|PARA|LIST|NOTE"/>
  </xsl:template>

  <!-- Second-Level Heading -->
  <xsl:template match="/ARTICLE/SECT/SECT">
      <h2> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </h2>
      <xsl:apply-templates select="SECT|PARA|LIST|NOTE"/>
  </xsl:template>

  <!-- Third-Level Heading -->
  <xsl:template match="/ARTICLE/SECT/SECT/SECT">
     <xsl:message terminate="yes">Error: Sections can only be nested 2
deep.</xsl:message>
  </xsl:template>

  <!-- Paragraph -->
  <xsl:template match="PARA">
      <p><xsl:apply-templates/></p>
  </xsl:template>

</xsl:stylesheet>
```

```
<html>
<body>

<h1 align="center">A Sample Article</h1>

<h1>The First Major Section


  </h1>
<p>This section will introduce a subsection.</p>
<h2>The Subsection Heading

     </h2>
<p>This is the text of the subsection.
       </p>

</body>
</html>
```

A Sample Article

The First Major Section

This section will introduce a subsection.

The Subsection Heading

This is the text of the subsection.

Error: Sections can only be nested 2 deep.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >
  <xsl:output method="html"/>
  <xsl:strip-space elements="SECT"/>

  <xsl:template match="/">
    <html><body>
       <xsl:apply-templates/>
    </body></html>
  </xsl:template>

  <xsl:template match="/ARTICLE/TITLE">
    <h1 align="center"> <xsl:apply-templates/> </h1>
  </xsl:template>

  <!-- Top Level Heading -->
  <xsl:template match="/ARTICLE/SECT">
      <h1> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </h1>
      <xsl:apply-templates select="SECT|PARA|LIST|NOTE"/>
  </xsl:template>

  <!-- Second-Level Heading -->
  <xsl:template match="/ARTICLE/SECT/SECT">
      <h2> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </h2>
      <xsl:apply-templates select="SECT|PARA|LIST|NOTE"/>
  </xsl:template>

  <!-- Third-Level Heading -->
  <xsl:template match="/ARTICLE/SECT/SECT/SECT">
     <xsl:message terminate="yes">Error: Sections can only be nested 2 
deep.</xsl:message>
  </xsl:template>

  <!-- Paragraph -->
  <xsl:template match="PARA">
      <p><xsl:apply-templates/></p>
  </xsl:template>

</xsl:stylesheet>
```

```
<html>
<body>

<h1 align="center">A Sample Article</h1>

<h1>The First Major Section
     </h1>
<p>This section will introduce a subsection.</p>
<h2>The Subsection Heading
       </h2>
<p>This is the text of the subsection.
       </p>

</body>
</html>
```

A Sample Article

The First Major Section

This section will introduce a subsection.

The Subsection Heading

This is the text of the subsection.

Error: Sections can only be nested 2 deep.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >
  <xsl:output method="html"/>
  <xsl:strip-space elements="SECT"/>

  <xsl:template match="/">
    <html><body>
       <xsl:apply-templates/>
    </body></html>
  </xsl:template>

  <xsl:template match="/ARTICLE/TITLE">
    <h1 align="center"> <xsl:apply-templates/> </h1>
  </xsl:template>

  <!-- Top Level Heading -->
  <xsl:template match="/ARTICLE/SECT">
      <h1> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </h1>
      <xsl:apply-templates select="SECT|PARA|LIST|NOTE"/>
  </xsl:template>

  <!-- Second-Level Heading -->
  <xsl:template match="/ARTICLE/SECT/SECT">
      <h2> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </h2>
      <xsl:apply-templates select="SECT|PARA|LIST|NOTE"/>
  </xsl:template>

  <!-- Third-Level Heading -->
  <xsl:template match="/ARTICLE/SECT/SECT/SECT">
     <xsl:message terminate="yes">Error: Sections can only be nested 2
deep.</xsl:message>
  </xsl:template>

  <!-- Paragraph -->
  <xsl:template match="PARA">
      <p><xsl:apply-templates/></p>
  </xsl:template>

  <!-- Text -->
  <xsl:template match="text()">
```

```
    <xsl:value-of select="normalize-space()"/>
  </xsl:template>

</xsl:stylesheet>
```

```
<html>
<body>
<h1 align="center">A Sample Article</h1>
<h1>The First Major Section</h1>
<p>This section will introduce a subsection.</p>
<h2>The Subsection Heading</h2>
<p>This is the text of the subsection.</p>
</body>
</html>
```

A Sample Article

The First Major Section

This section will introduce a subsection.

The Subsection Heading

This is the text of the subsection.

The First Major Section This section will introduce a subsection. The Subsection Heading This is the text of the subsection. The Second Major Section This section adds a LIST and a NOTE. Here is the LIST: Pears Grapes And here is the NOTE: Don't forget to go to the hardware store on your way to the grocery!

```
<?xml version="1.0"?>
<ARTICLE>
  <TITLE>A Sample Article</TITLE>
  <SECT>The First Major Section
     <PARA>This section will introduce a subsection.</PARA>
     <SECT>The Subsection Heading
       <PARA>This is the text of the subsection.
       </PARA>
     </SECT>
  </SECT>
  <SECT>The Second Major Section
     <PARA>This section adds a LIST and a NOTE.
     </PARA>
     <PARA>Here is the LIST:
       <LIST type="ordered">
           <ITEM>Pears</ITEM>
           <ITEM>Grapes</ITEM>
       </LIST>
     </PARA>
     <PARA>And here is the NOTE:
       <NOTE>Don't forget to go to the hardware store on your
             way to the grocery!
       </NOTE>
     </PARA>
  </SECT>
</ARTICLE>
```

Error: Sections can only be nested 2 deep.

-

Note:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >
  <xsl:output method="html"/>
  <xsl:strip-space elements="SECT"/>

  <xsl:template match="/">
    <html><body>
       <xsl:apply-templates/>
    </body></html>
  </xsl:template>

  <xsl:template match="/ARTICLE/TITLE">
    <h1 align="center"> <xsl:apply-templates/> </h1>
  </xsl:template>

  <!-- Top Level Heading -->
  <xsl:template match="/ARTICLE/SECT">
      <h1> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </h1>
      <xsl:apply-templates select="SECT|PARA|LIST|NOTE"/>
  </xsl:template>

  <!-- Second-Level Heading -->
  <xsl:template match="/ARTICLE/SECT/SECT">
      <h2> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </h2>
      <xsl:apply-templates select="SECT|PARA|LIST|NOTE"/>
  </xsl:template>

  <!-- Third-Level Heading -->
  <xsl:template match="/ARTICLE/SECT/SECT/SECT">
     <xsl:message terminate="yes">Error: Sections can only be nested 2
deep.</xsl:message>
  </xsl:template>

  <!-- Paragraph -->
  <xsl:template match="PARA">
      <!-- MODIFIED -->
      <!-- OLD: <p><xsl:apply-templates/></p> -->
      <p> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </p>
      <xsl:apply-templates select="PARA|LIST|NOTE"/>
  </xsl:template>
  <!-- Text -->
  <xsl:template match="text()">
    <xsl:value-of select="normalize-space()"/>
  </xsl:template>

  <!-- LIST  -->
  <xsl:template match="LIST">
    <xsl:if test="@type='ordered'">
      <ol>
      <xsl:apply-templates/>
      </ol>
    </xsl:if>
    <xsl:if test="@type='unordered'">
      <ul>
      <xsl:apply-templates/>
      </ul>
    </xsl:if>
  </xsl:template>

  <!-- list ITEM -->
  <xsl:template match="ITEM">
    <li><xsl:apply-templates/>
    </li>
  </xsl:template>

  <xsl:template match="NOTE">
    <blockquote><b>Note:</b><br/>
      <xsl:apply-templates/>
    </blockquote>
  </xsl:template>

</xsl:stylesheet>
```

```
<html>
<body>
<h1 align="center">A Sample Article</h1>
<h1>The First Major Section</h1>
<p>This section will introduce a subsection.</p>
<h2>The Subsection Heading</h2>
<p>This is the text of the subsection.</p>
<h1>The Second Major Section</h1>
<p>This section adds a LIST and a NOTE.</p>
<p>Here is the LIST:</p>
<ol>
<li>Pears</li>
<li>Grapes</li>
</ol>
<p>And here is the NOTE:</p>
<blockquote>
<b>Note:</b>
<br>Don't forget to go to the hardware store on your way to the grocery!</blockquote>
</body>
</html>
```

A Sample Article

The First Major Section

This section will introduce a subsection.

The Subsection Heading

This is the text of the subsection.

The Second Major Section

This section adds a LIST and a NOTE.

Here is the LIST:

1. Pears
2. Grapes

And here is the NOTE:

> **Note:**
> Don't forget to go to the hardware store on your way to the grocery!

The First Major Section This section will introduce a subsection. The Subsection Heading This is the text of the subsection. The Second Major Section This section adds a LIST and a NOTE. Here is the LIST: Pears Grapes And here is the NOTE: Don't forget to go to the hardware store on your way to the grocery! The *Third* Major Section In addition to the inline tag in the heading, this section defines the term inline, which literally means "no line break". It also adds a simple link to the main page for the Java platform (http://java.sun.com), as well as a link to the XML page.

```
<?xml version="1.0"?>
<ARTICLE>
  <TITLE>A Sample Article</TITLE>
  <SECT>The First Major Section
     <PARA>This section will introduce a subsection.</PARA>
     <SECT>The Subsection Heading
       <PARA>This is the text of the subsection.
       </PARA>
     </SECT>
  </SECT>
  <SECT>The Second Major Section
     <PARA>This section adds a LIST and a NOTE.
     </PARA>
     <PARA>Here is the LIST:
       <LIST type="ordered">
           <ITEM>Pears</ITEM>
           <ITEM>Grapes</ITEM>
       </LIST>
     </PARA>
     <PARA>And here is the NOTE:
       <NOTE>Don't forget to go to the hardware store on your
             way to the grocery!
       </NOTE>
     </PARA>
  </SECT>
  <SECT>The <I>Third</I> Major Section
     <PARA>In addition to the inline tag in the heading, this section
           defines the term <DEF>inline</DEF>, which literally means
           "no line break". It also adds a simple link to the main page
           for the Java platform (<LINK>http://java.sun.com</LINK>),
           as well as a link to the
           <LINK target="http://java.sun.com/xml">XML</LINK> page.
     </PARA>
  </SECT>
</ARTICLE>
```

Error: Sections can only be nested 2 deep.

-

Note:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >
  <xsl:output method="html"/>
  <xsl:strip-space elements="SECT"/>

  <xsl:template match="/">
    <html><body>
       <xsl:apply-templates/>
    </body></html>
  </xsl:template>

  <xsl:template match="/ARTICLE/TITLE">
    <h1 align="center"> <xsl:apply-templates/> </h1>
  </xsl:template>

  <!-- Top Level Heading -->
  <xsl:template match="/ARTICLE/SECT">
      <h1> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </h1>
      <xsl:apply-templates select="SECT|PARA|LIST|NOTE"/>
  </xsl:template>

  <!-- Second-Level Heading -->
  <xsl:template match="/ARTICLE/SECT/SECT">
      <h2> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </h2>
      <xsl:apply-templates select="SECT|PARA|LIST|NOTE"/>
  </xsl:template>

  <!-- Third-Level Heading -->
  <xsl:template match="/ARTICLE/SECT/SECT/SECT">
     <xsl:message terminate="yes">Error: Sections can only be nested 2
deep.</xsl:message>
  </xsl:template>

  <!-- Paragraph -->
  <xsl:template match="PARA">
    <p> <xsl:apply-templates select="text()|B|I|U|DEF|LINK"/> </p>
    <xsl:apply-templates select="PARA|LIST|NOTE"/>
  </xsl:template>

  <!-- Text -->
<!--
  <xsl:template match="text()">
    <xsl:value-of select="normalize-space()"/>
  </xsl:template>
-->

  <!-- LIST  -->
  <xsl:template match="LIST">
```

```
    <xsl:if test="@type='ordered'">
      <ol>
      <xsl:apply-templates/>
      </ol>
    </xsl:if>
    <xsl:if test="@type='unordered'">
      <ul>
      <xsl:apply-templates/>
      </ul>
    </xsl:if>
  </xsl:template>

  <!-- list ITEM -->
  <xsl:template match="ITEM">
    <li><xsl:apply-templates/>
    </li>
  </xsl:template>

  <xsl:template match="NOTE">
    <blockquote><b>Note:</b><br/>
      <xsl:apply-templates/>
    </blockquote>
  </xsl:template>

  <xsl:template match="DEF">
     <i> <xsl:apply-templates/> </i>
  </xsl:template>

  <xsl:template match="B|I|U">
     <xsl:element name="{name()}">
       <xsl:apply-templates/>
     </xsl:element>
  </xsl:template>

  <xsl:template match="LINK">
    <xsl:if test="@target">
      <!--Target attribute specified.-->
      <xsl:call-template name="htmLink">
        <xsl:with-param name="dest" select="@target"/>  <!--Destination = attribute
value-->
      </xsl:call-template>
    </xsl:if>

    <xsl:if test="not(@target)">
      <!--Target attribute not specified.-->
      <xsl:call-template name="htmLink">
        <xsl:with-param name="dest">
          <xsl:apply-templates/>  <!--Destination value = text of node-->
        </xsl:with-param>
      </xsl:call-template>
    </xsl:if>
  </xsl:template>
  <!-- A named template that constructs an HTML link -->
  <xsl:template name="htmLink">
    <xsl:param name="dest" select="UNDEFINED"/> <!--default value-->
    <xsl:element name="a">
      <xsl:attribute name="href">
        <xsl:value-of select="$dest"/> <!--link target-->
      </xsl:attribute>
      <xsl:apply-templates/> <!--name of the link from text of node-->
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

```
<html>
<body>

<h1 align="center">A Sample Article</h1>

<h1>The First Major Section
     </h1>
<p>This section will introduce a subsection.</p>
<h2>The Subsection Heading
       </h2>
<p>This is the text of the subsection.
       </p>

<h1>The Second Major Section
     </h1>
<p>This section adds a LIST and a NOTE.
     </p>
<p>Here is the LIST:
       </p>
<ol>
<li>Pears</li>
<li>Grapes</li>
</ol>
<p>And here is the NOTE:
       </p>
<blockquote>
<b>Note:</b>
<br>Don't forget to go to the hardware store on your
             way to the grocery!
       </blockquote>

<h1>The <I>Third</I> Major Section
     </h1>
<p>In addition to the inline tag in the heading, this section
           defines the term <i>inline</i>, which literally means
           "no line break". It also adds a simple link to the main page
           for the Java platform (<a
href="http://java.sun.com">http://java.sun.com</a>),
           as well as a link to the
           <a href="http://java.sun.com/xml">XML</a> page.
     </p>

</body>
</html>
```

A Sample Article

The First Major Section

This section will introduce a subsection.

The Subsection Heading

This is the text of the subsection.

The Second Major Section

This section adds a LIST and a NOTE.

Here is the LIST:

1. Pears
2. Grapes

And here is the NOTE:

> **Note:**
> Don't forget to go to the hardware store on your way to the grocery!

The *Third* Major Section

In addition to the inline tag in the heading, this section defines the term *inline*, which literally means "no line break". It also adds a simple link to the main page for the Java platform (http://java.sun.com), as well as a link to the XML page.

```java
/*
 * @(#)FilterChain.java 1.9 98/11/10
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 */

import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;
import org.xml.sax.XMLFilter;

import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerConfigurationException;

import javax.xml.transform.sax.SAXTransformerFactory;
import javax.xml.transform.sax.SAXSource;
import javax.xml.transform.sax.SAXResult;

import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;

import java.io.*;

public class FilterChain
{
  public static void main (String argv [])
  {
    if (argv.length != 3) {
      System.err.println ("Usage: java FilterChain stylesheet1 stylesheet2 xmlfile");
      System.exit (1);
    }

    try {
      // Read the arguments
      File stylesheet1 = new File(argv[0]);
      File stylesheet2 = new File(argv[1]);
      File datafile    = new File(argv[2]);

      // Set up the input stream
      BufferedInputStream bis = new BufferedInputStream(new FileInputStream(datafile));
      InputSource input = new InputSource(bis);

      // Set up to read the input file
      SAXParserFactory spf = SAXParserFactory.newInstance();
      SAXParser parser = spf.newSAXParser();
      XMLReader reader = parser.getXMLReader();

      // Create the filters
      // --SAXTransformerFactory is an interface
      // --TransformerFactory is a concrete class
      // --TransformerFactory actually returns a SAXTransformerFactory instance
      // --We didn't care about that before, because we didn't use the
      // --SAXTransformerFactory extensions. But now we do, so we cast the result.
      SAXTransformerFactory stf =
        (SAXTransformerFactory) TransformerFactory.newInstance();
      XMLFilter filter1 = stf.newXMLFilter(new StreamSource(stylesheet1));
      XMLFilter filter2 = stf.newXMLFilter(new StreamSource(stylesheet2));

      // Wire the output of the reader to filter1
      // and the output of filter1 to filter2
      // --A filter is a kind of reader
      // --Setting the parent sets the input reader
      // --Since a filter is a reader, the "parent" could be another filter
      filter1.setParent(reader);
      filter2.setParent(filter1);

      // Set up the output stream
      StreamResult result = new StreamResult(System.out);

      // Set up the transformer to process the SAX events generated
      // by the last filter in the chain
      Transformer transformer = stf.newTransformer();
      SAXSource transformSource = new SAXSource(filter2, input);
      transformer.transform(transformSource, result);
    }
    catch (TransformerConfigurationException tce) {
      // Error generated by the parser
      System.out.println ("\n** Transformer Factory error");
      System.out.println("   " + tce.getMessage() );

      // Use the contained exception, if any
      Throwable x = tce;
      if (tce.getException() != null)
         x = tce.getException();
      x.printStackTrace();
    }
    catch (TransformerException te) {
      // Error generated by the parser
      System.out.println ("\n** Transformation error");
      System.out.println("   " + te.getMessage() );

      // Use the contained exception, if any
      Throwable x = te;
      if (te.getException() != null)
         x = te.getException();
      x.printStackTrace();
    }
    catch (SAXException sxe) {
      // Error generated by this application
      // (or a parser-initialization error)
      Exception  x = sxe;
      if (sxe.getException() != null)
         x = sxe.getException();
      x.printStackTrace();
    }
    catch (ParserConfigurationException pce) {
      // Parser with specified options can't be built
      pce.printStackTrace();
    }
    catch (IOException ioe) {
      // I/O error
      ioe.printStackTrace();
    }

  } // main

}
```

```
<?xml version="1.0"?>
<Article>
  <ArtHeader>
    <Title>Title of my (Docbook) article</Title>
  </ArtHeader>
  <Sect1>
    <Title>Title of Section 1.</Title>
    <Para>This is a paragraph.</Para>
  </Sect1>
</Article>
```

```xml
<?xml version="1.0"?>
<Article>
  <ArtHeader>
    <Title>Title of my (Docbook) article</Title>
  </ArtHeader>
  <Sect1>
    <Title>Title of Section 1.</Title>
    <Para>This is a paragraph.</Para>
  </Sect1>
</Article>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  >
  <!-- XML output! -->
  <xsl:output method="xml"/>

  <xsl:template match="/">
    <ARTICLE>
       <xsl:apply-templates/>
    </ARTICLE>
  </xsl:template>

  <!-- Lower level titles strip out the element tag -->

  <!-- Top-level title -->
  <xsl:template match="/Article/ArtHeader/Title">
    <TITLE> <xsl:apply-templates/> </TITLE>
  </xsl:template>

  <xsl:template match="//Sect1">
      <SECT><xsl:apply-templates/></SECT>
  </xsl:template>

  <!-- Case-change -->
  <xsl:template match="Para">
      <PARA><xsl:apply-templates/></PARA>
  </xsl:template>

</xsl:stylesheet>
```

```
<html>
<body>
<h1 align="center">Title of my (Docbook) article</h1>
<h1>Title of Section 1.</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Title of my (Docbook) article

Title of Section 1.

This is a paragraph.

Bios for Contributing Authors

Eric Armstrong

Links to Author's Work	Bio	Acknowledgments
XML	**Eric Armstrong** contracts at Sun Microsystems to help spread Java and XML technology. He has been programming and writing professionally since before there were personal computers. He has written for JavaWorld and is the author of The JBuilder 2 Bible.	The XML tutorial could not have been written without the excellent explanations provided by David Brownell, the original team lead. Rajiv Mordani, Edwin Goei, and current project lead James Duncan Davidson also proved enormously valuable, guiding me through the APIs while providing both technical advice and design feedback.