



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>


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™ 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.

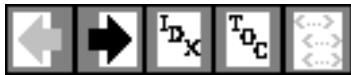
 [Part IV: Using XSLT](#) shows how the XSL transformation package can be used to write out a DOM as XML, convert arbitrary data to XML by creating a SAX parser, and convert XML data into a different format.

 [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:

- [A Quick Introduction to XML](#)
- [Writing a Simple XML File](#)
- [Substituting and Inserting Text](#)
- [Defining a Document Type](#)
- [Defining Attributes and Entities](#)
- [Referencing Binary Entities](#)
- [Defining Parameter Entities](#)
- [Designing an XML Document](#)



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

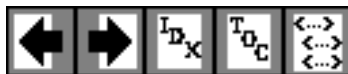
Part I. Understanding XML and the Java XML APIs

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.





<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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?](#)

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" (. . .), 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.

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](#)

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 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 Reusability

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".

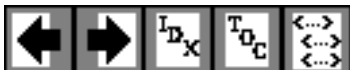
XML's text-based format could also make it easier to transfer objects between applications written in different languages. For all of these reasons, XML-based archiving is likely to become an important force in the not-too-distant future.

Summary

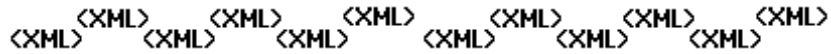
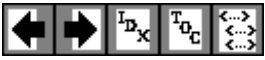
XML is pretty simple, and very flexible. It has many uses yet to be discovered -- we are just beginning to scratch the surface of its potential. It is the foundation for a great many standards yet to come, providing a common language that different computer systems can use to exchange data with one another. 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!

Basic Standards

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](#)

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

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.

- [cXML](#)
- [CBL](#)

Glossary Terms

[DTD](#), [entity](#), [prolog](#)

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

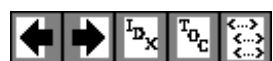
Common Business Library

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

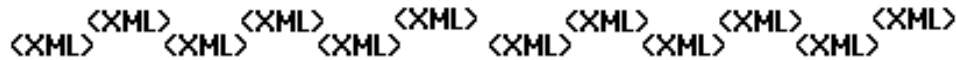
For more information on CBL and a variety of other initiatives that work together to enable eCommerce applications, see http://www.commerce.net/projects/currentprojects/eco/wg/eCo_Framework_Specifications.html.

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:

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

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](#)

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.

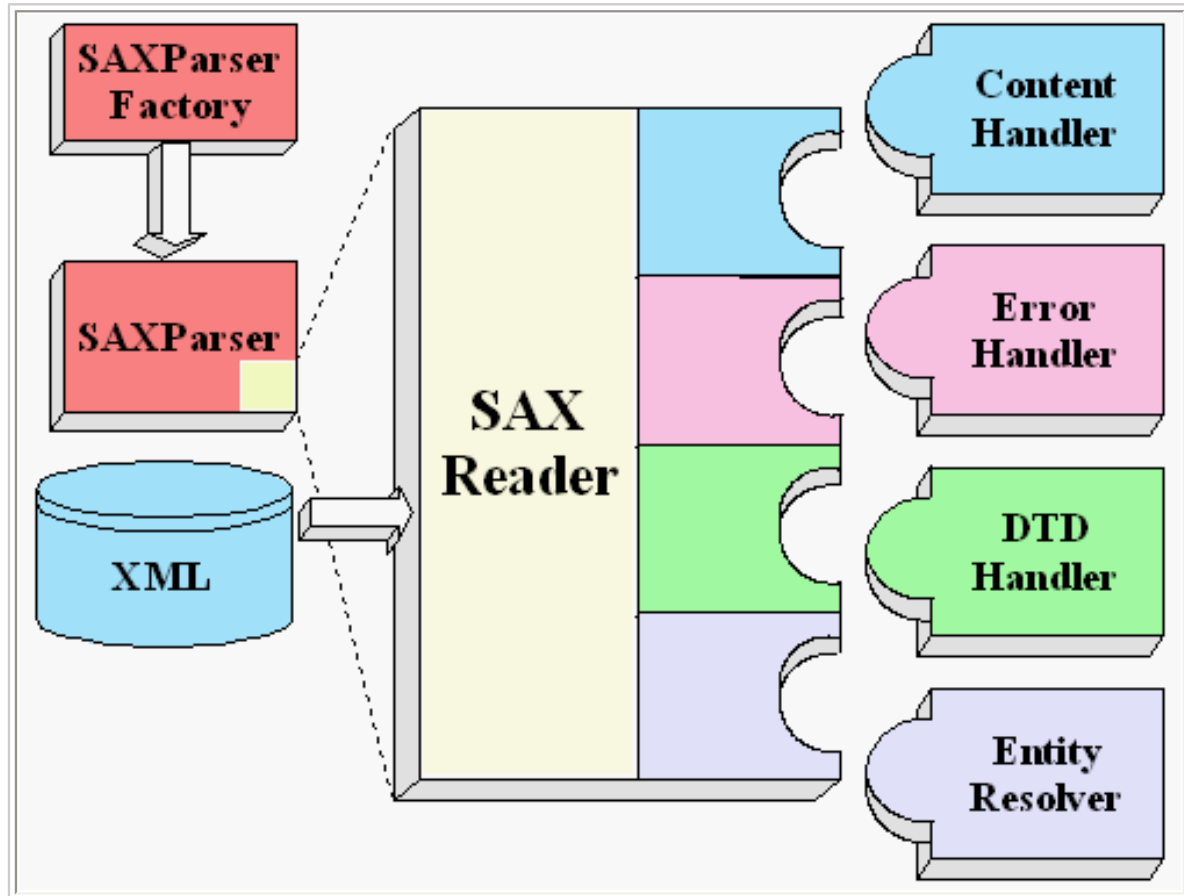
Finally, the XSLT APIs defined in `javax.xml.transform` let you write XML data to a file or convert it into other forms. And, as you'll see in the XSLT section, of this tutorial, you can even use it in conjunction with the SAX APIs to convert legacy data to XML.

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

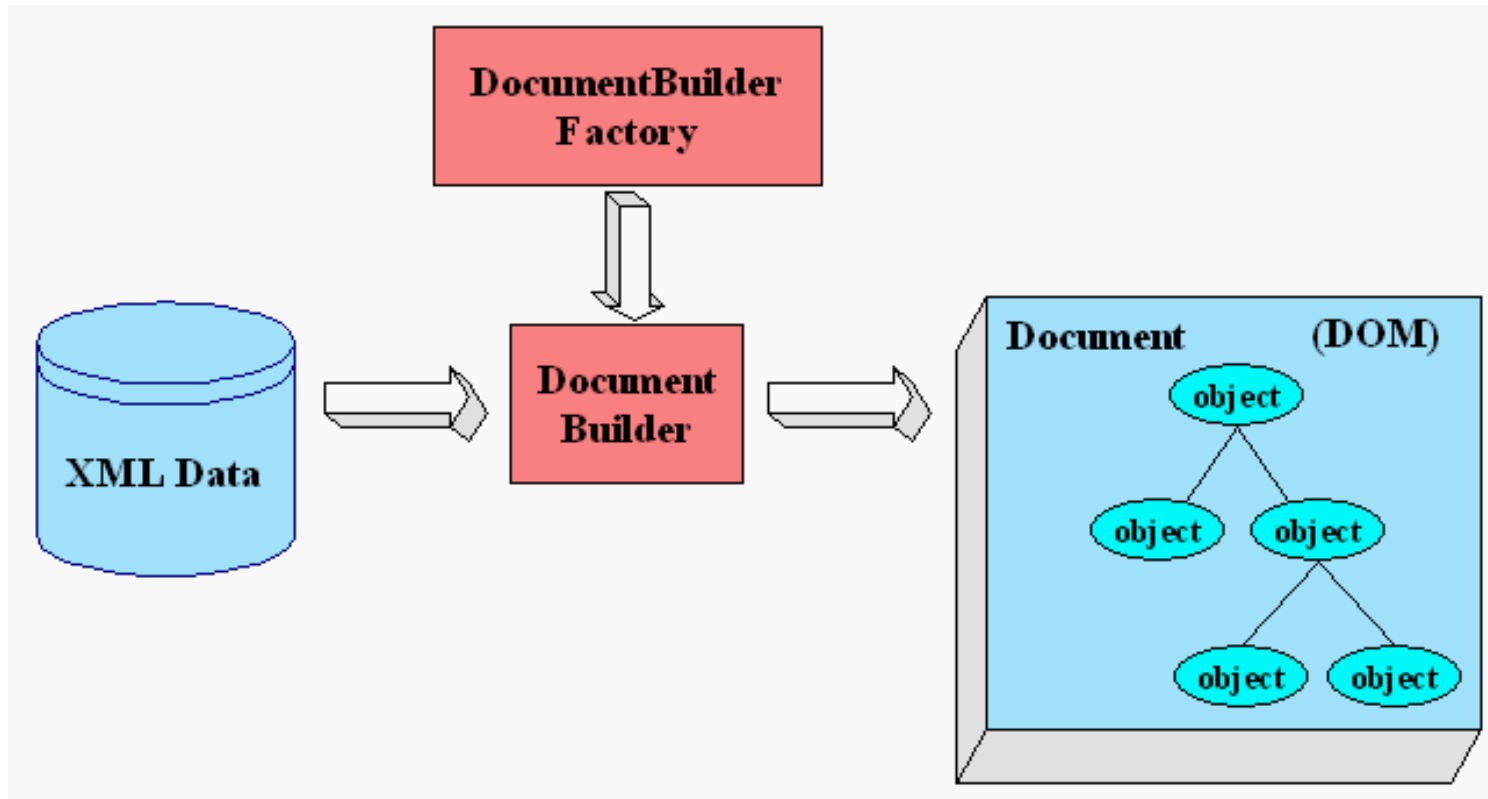
The SAX parser is defined in the following packages.

<i>Package</i>	<i>Description</i>
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:

<i>Package</i>	<i>Description</i>
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 <code>javax.xml.parsers</code> system property, which can be set from the command line or overridden when invoking the <code>newInstance</code> method. This package also defines the <code>ParserConfigurationException</code> class for reporting errors.

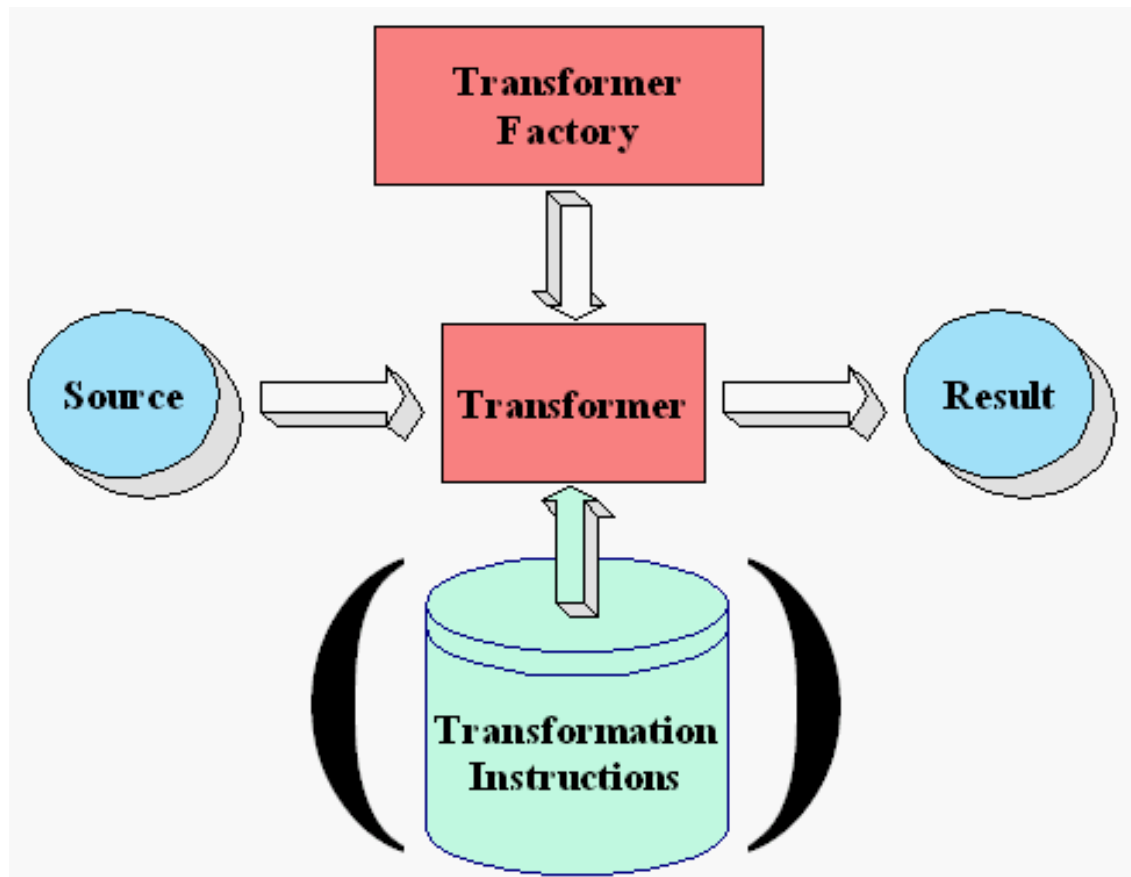
The XML Style Sheet Translation (XSLT) APIs

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:

<i>Package</i>	<i>Description</i>
----------------	--------------------

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.

<i>JAR file</i>	<i>Packages</i>	<i>Contents</i>
jaxp.jar	<ul style="list-style-type: none"> • javax.xml.parsers • javax.xml.transform <ul style="list-style-type: none"> ◦ javax.xml.transform.dom ◦ javax.xml.transform.sax ◦ javax.xml.transform.stream 	Interfaces
crimson.jar	<ul style="list-style-type: none"> • org.xml.sax <ul style="list-style-type: none"> ◦ 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.

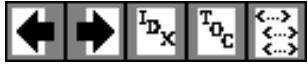
Browse the [Examples](#)

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.

<i>Example</i>	<i>Description</i>
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.



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

4. Designing an XML Data Structure

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

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

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](#)

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 `Best` 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.

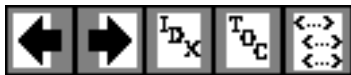
Note:

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.



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



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](#) <#> [_](#)

A

ANY

[DTD element value](#)

[archiving](#)

[use of XML for](#)

ATTLIST

[DTD tag, defining an attribute list](#)

[attribute](#)

[and tags](#)

[adding to an element, naming](#)

[defining in the DTD](#)

[specification of, with ATTLIST tag](#)

[vs element design decision](#)

B

[binary entity](#)

See [unparsed entity](#).

[binding](#)

[use of XML for](#)

C

CBL

[XML-based standard](#)

CDATA

[special XML tag for handling text with XML-style syntax](#)

[special DTD qualifier used for specifying attribute values](#)

[need for a LexicalEventListener when generating XML output](#)

[using a LexicalEventListener to echo in XML output](#)

[echoing of, with a LexicalEventListener in the SAX echo app](#)

[DTD attribute type](#)

characters

[special characters, handling of](#)

[character references](#)

[vs. ignorable whitespace](#)

character encodings

See [encodings](#)

command line

[use of environment variable to select validating or nonvalidating parser](#)

command scripts

[for compiling and running Java XML apps](#)

comment

[Comments in XML Files](#)

[writing an XML file](#)

[using a LexicalEventListener to echo in XML output](#)

[echoing of, with a LexicalEventListener in the SAX echo app](#)

compiling

[of SAX echo app](#)

See Also: [command scripts](#)

conditional sections

[in a DTD, controllable with parameter entities](#)

consumer/provider

[attributes vs. elements, stylistic choice](#)

container/contents

[attributes vs. elements, stylistic choice](#)

[content](#)

[and the XML prolog](#)

ContentHandler interface

[handling document events](#)

[SAX API](#)

[SAXExceptions thrown](#)

[identifying a document's location](#)

[processing instructions, handling of](#)

[supplied to parser in SAX echo app](#)

[extended by LexicalEventListener](#)

copyright symbol

[entity definition](#)

cXML

[XML-based standard](#)

D

[data](#)

[contrasted with "document"](#)

[data elements](#)

[identified by tags](#)

[normalization of](#)

design of XML documents

[attributes vs elements](#)

[attributes vs elements, example of](#)

[consumer/provider](#)

[container/contents](#)

[visibility heuristic](#)

[DDP](#)

[use of XML for](#)

declaration

[The XML Prolog](#)
[creating an XML file](#)

DefaultHandler class

[implements SAX APIs](#)
[use of, in Echo app](#)
[extending of, to implement the ErrorHandler interface](#)

DOCTYPE tag

[use of, to specify the DTD](#)

document

[contrasted with "data"](#)
[document elements](#)

DOM

[Java XML APIs, overview](#)
[Document Object Model](#)
[XML and Related Specs](#)
[w3c specification](#)

double quotes (curly quotes)

[entity definition](#)

DTD

[compared to schemas](#)
[creating](#)
[defining attributes](#)
[defining attributes, ATTLIST tag](#)
[defining elements \(element declarations\)](#)
[defining entities](#)
[defining nested elements](#)
[defining the root element](#)
[defining text](#)
[effect on definition of empty elements](#)
[limitations of](#)

- [nonvalidating parser, effect on](#)
- [normalization of](#)
- [parameter entities, creating and referencing of](#)
- [parsing a parameterized DTD](#)
- [referencing the DTD](#)
- [required for generation of non-fatal errors](#)
- [required for generation of warnings](#)
- [required to use validating parser](#)
- [special element value: ANY](#)
- [special element value: EMPTY](#)
- [specifying with the DOCTYPE tag](#)
- [use of existing, if possible](#)
- [use of, for binding](#)
- [w3c specification](#)
- [XML and Related Specs](#)

DTDEventListener interface

- [Java XML API](#)
- [using startDTD and endDTD methods to identify DTD processing](#)

DTDHandler interface

- [SAX API](#)
- [using the DTDHandler API](#)

E

[element](#)

- [defined by tags](#)
- [defining in the DTD](#)
- [special DTD values ANY and EMPTY](#)
- [naming of, adding attributes to](#)
- [nesting of](#)
- [nested elements, adding](#)
- [vs attribute design decision](#)

empty element

- [adding of](#)
- [as further defined by the DTD](#)

EMPTY

[DTD element value](#)

encodings

[Java's encoding schemes](#)

[setting up I/O in SAX echo app](#)

entity

definable in [The XML Prolog](#)

[defining in the DTD](#)

[predefined entities for special characters](#)

[referenced in an XML Document](#)

[summary of entity types](#)

[DTD attribute type](#)

[useful entity definitions](#)

entity reference

[to predefined entities for special characters](#)

[use of, in an XML Document](#)

[echoing of, in the SAX echo app](#)

[referencing external entities](#)

[use of for binary entities \(not recommended\)](#)

ENTITY

[DTD](#) tag

ENTITIES

[DTD attribute type](#)

EntityResolver interface

[SAX API](#)

[using the EntityResolver API](#)

environment variable

[use of, to select validating or nonvalidating parser](#)

error conditions

[Handling Errors with the Non-Validating Parser](#)

fatal error

[introducing](#)

[\(non-fatal\) error](#)

[understanding](#)

[handling, mechanisms](#)

[experimenting with validation errors in the SAX echo app](#)

[handling of, in the validating parser](#)

[warning](#)

[DTD required for, handling of](#)

[DTD warnings that could occur](#)

ErrorHandler interface

[SAX API](#)

[use of, for handling non-fatal errors](#)

examples

[description of, in Java XML release](#)

exceptions

ParserConfigurationException

[handling of, in a SAX app](#)

IOException

[wrapping in a SAX exception](#)

SAXException

[thrown by DocumentHandler methods](#)

[handling of](#)

SAXParseException

[basic handling of](#)

[improved handling of](#)

[delivered to ErrorHandler methods](#)

[external entity](#)

[summary of entity types](#)

[referencing](#)

[echoing, with the SAX echo app](#)

[use of, for normalizing an XML structure](#)

[external subset](#)

[referencing the DTD](#)

F

fatal error

See [error conditions](#)

FIXED

[attribute specification in the DTD](#)

flow objects

[defined by XSL](#)

formatting objects

[defined by XSL](#)

G

[general entity](#)

[summary of entity types](#)

H

[HTML](#)

[HTML-style text, adding of](#)

[inline reusability -- none, vs. XML](#)

[linking, compared to XML](#)

[tags similar to XML](#)

[vs. stylability of XML documents](#)

[reusing a DTD that defines HTML-style tags in XML](#)

I

ICE

[XML-based standard](#)

ID

[DTD attribute type](#)

identity transform

[defined](#)

IDREF

[DTD attribute type](#)

IDREFS

[DTD attribute type](#)

IMPLIED

[attribute specification in the DTD](#)

InputSource class

[used in setting up the SAX echo app](#)

IOException

wrapping in a SAX exception: see [exceptions](#)

J

K

L

LexicalHandler interface

[must use, else comments do not appear in SAX echo app](#)

[how it works](#)

[working with a LexicalHandler](#)

line endings

[normalization of, to NL \(\n\)](#)

[local subset](#)

[referencing the DTD](#)

M

MathML

[XML-based standard](#)

methods

comment

[LexicalEventListener interface](#)

characters

[handling document events](#)

[vs. ignorableWhitespace](#)

ignorableWhitespace

[documents vs. data, DTD required](#)

[vs. characters method](#)

makeParser

[setting up the SAX echo app](#)

notationDecl

[the DTDHandler API](#)

processingInstruction

[processing instructions, handling of](#)

resolveEntity

[the EntityResolver API](#)

setDocumentLocator

[identifying a document's location](#)

[order of, with respect to startDocument](#)

startCDATA / endCDATA

[LexicalHandler interface](#)

startDocument / endDocument

[explanation of use in the SAX echo app](#)

[handling document events](#)

[order of, with respect to setDocumentLocator](#)

startDtd / endDtd

[using to identify DTD processing with a DtdEventListener](#)

startElement / endElement

[handling document events](#)

startParsedEntity / endParsedEntity

[LexicalHandler interface](#)

unparsedEntityDecl

[the DTDHandler API](#)

MIME data types

[used to reference a binary entity](#)

mixed-content model

[defining in the DTD](#)

N

Namespace

[XML and Related Specs](#)

[used to reference a binary entity](#)

[w3c specification](#)

NDATA

[defining an unparsed entity with](#)

NMTOKEN

[DTD attribute type](#)

NMTOKENS

[DTD attribute type](#)

nonvalidating parser

See [parser](#)

normalization

[Normalizing Data](#)

[Normalizing DTDs](#)

[normalization of line endings to NL \(\n\)](#)

notation

[use of for binary entities \(not recommended\)](#)

NOTATION

[DTD attribute type](#)

[use of for binary entities \(not recommended\)](#)

[processing of, using the DTDHandler API](#)

O

OASIS

[comments on design of XML documents](#)
[repository for DTDs](#)

output

[compressing of, in SAX echo app](#)
[spacing of, in SAX echo app](#)
[writing of, in SAX echo app](#)

P

packages

[org.xml.sax](#)
[org.xml.sax.helpers](#)
[org.w3c.dom](#)
[imported into SAX echo app](#)

parameter entity

[creating and referencing in the DTD](#)
[for reusing an existing DTD definition](#)
[summary of entity types](#)
[use of, to control conditional sections](#)

parsed entity

[summary of entity types](#)
[using a LexicalEventListener to echo in XML output](#)
[echoing of, with a LexicalEventListener in the SAX echo app](#)

parser

[SAX Parser](#)
[echoing an XML file with](#)
[effect of DTD on, overview of](#)
[DTD's Effect on the Nonvalidating Parser](#)
[selected using an environment variable](#)

Parser interface

[SAX Parser](#)

ParserConfigurationException

See [exceptions](#)

ParserFactory class

[SAX Parser](#)

[imported into SAX echo app](#)

[used in setting up the SAX echo app](#)

PCDATA

[DTD keyword](#)

[processing instruction](#)

[declaration of](#)

[handling of](#)

[prolog](#)

[The XML Prolog](#)

public ID

See [URN](#)

Q

R

REQUIRED

[attribute specification in the DTD](#)

[RDF](#)

[used for traditional data processing](#)

[XML and Related Specs](#)

[w3c specification](#)

[RDF Schema](#)

[XML and Related Specs](#)

[w3c specification](#)

reference

See [entity reference](#).

RELAX

[schema standard](#)

root

[writing an XML file](#)

[defining in the DTD](#)

running

[of SAX echo app](#)

See Also: [command scripts](#)

S

SAX

[echoing an XML file with](#)

[Java XML APIs, overview](#)

[SAX Parser](#)

[serial access with](#)

[XML and Related Specs](#)

[w3c specification](#)

SAXException

see [exceptions](#)

SAXParseException

see [exceptions](#)

schema

[use of, for binding](#)

[standards](#)

schematron

[schema standard](#)

scripts

See: [command scripts](#)

[SGML](#)

...

SMIL

[XML-based standard](#)

SOX

[schema standard](#)

SVG

[XML-based standard](#)

system ID

See [URL](#)

T

[tag](#)

[used to identify data](#)

[Tags and Attributes](#)

[Empty Tags](#)

[naming of](#)

text

[handling text with XML-style syntax](#)

topic maps

[knowledge standard](#)

trademark symbols

[entity definitions](#)

TREX

[schema standard](#)

U

unicode

...

unparsed entity

[summary of entity types](#)

[referencing a binary entity](#)

[defining with the NDATA keyword](#)

URI

[use of, when specifying a parameter entity](#)

See Also: [URL](#) and [URN](#)

URL

[identifying the document's location in the SAX echo app](#)

[specifying an external entity using a SYSTEM identifier](#)

[use of, to specify the DTD](#)

URN

[identifying the document's location in the SAX echo app](#)

[specifying an external entity using a PUBLIC identifier](#)

[use of EntityResolver to convert to a URL](#)

US-ASCII

[Java's encoding schemes](#)

[setting up I/O in SAX echo app](#)

UTF-8

[Java's encoding schemes](#)

[setting up I/O in SAX echo app](#)

UTF-16

[Java's encoding schemes](#)

[setting up I/O in SAX echo app](#)

V

valid

[as determined by a DTD](#)

validating parser

- [effect of DTD on](#)
- [error handling in](#)
- [generation of non-fatal errors, from DTD](#)
- [selected using an environment variable](#)
- [use of, in SAX echo app](#)

ValidatingParser interface
[SAX Parser](#)

visibility heuristic
[attributes vs. elements, stylistic choice](#)
[example of](#)

W

w3c

[XML and Related Specs.](#)

warning
See [error conditions](#)

well-formed

- [with respect to empty Tags](#)
- [nesting of tags](#)

whitespace
[tracking ignorable whitespace](#)

X

XHTML

- [XML and Related Specs](#)
- [w3c specification](#)
- [reuse of XHTML DTD in the SAX echo app](#)

XLink

[XML and Related Specs](#)

[w3c specification](#)

XLL

[XML and Related Specs](#)

[w3c specification](#)

XML

[A Quick Introduction to XML](#)

[archiving, using XML](#)

[attributes](#)

[binary entitites](#)

[binding, using XML](#)

[comments](#)

[designing an XML document](#)

[Document Type Definition](#)

[entities](#)

HTML, comparison with

[inline reusability](#)

[linking](#)

[similarity of tags](#)

[stylability of documents](#)

[parameter entities](#)

[processing instructions](#)

[prolog](#)

[text, substituting of](#)

[Writing a Simple XML File](#)

[XML and Related Specs](#)

XML Schema

[schema standard](#)

Xpath

[XSLT and,](#)

XPointer

[XML and Related Specs](#)

[w3c specification](#)

XSL

[used for traditional data processing](#)
[stylability of XML documents](#)
[XML and Related Specs](#)
[XSLT and,](#)
[w3c specification](#)

XSL-FO

[part of XSL](#)

XSLT

[basic standard](#)
[overview of APIs](#)
[part of XSL](#)
[using](#)

XTM

[XML Topic Maps](#)

Y

Z

_ (non-alpha)

[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](#)



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

Working with XML: Table of Contents

[Part I: Understanding XML and the Java XML APIs](#)

[1. A Quick Introduction to XML](#)

[What is XML?](#)

[Why is XML Important?](#)

[What can you do with XML?](#)

[2. XML and Related Specs: Digesting the Alphabet Soup](#)

[Basic Standards](#)

[Schema Standards](#)

[Linking and Presentation Standards](#)

[Knowledge Standards](#)

[Standards that Build on XML](#)

[Extended Document Standards](#)

[eCommerce Standards](#)

[3. An Overview of the APIs](#)

[The JAXP APIs](#)

[An Overview of the Packages](#)

[The Simple API for XML \(SAX\) APIs](#)

[The Document Object Model \(DOM\) APIs](#)

[The XML Style Sheet Translation \(XSLT\) APIs](#)

[Overview of the JAR Files](#)

[Where Do You Go from Here?](#)

[4. Designing an XML Data Structure](#)

[Saving Yourself Some Work](#)

[Attributes and Elements](#)

[Normalizing Data](#)

[Normalizing DTDs](#)

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

1. Writing a Simple XML File

[Creating the File](#)

[Writing the Declaration](#)

[Adding a Comment](#)

[Defining the Root Element](#)

[Adding Attributes to an Element](#)

[Adding Nested Elements](#)

[Adding HTML-Style Text](#)

[Adding an Empty Element](#)

[The Finished Product](#)

2a. Echoing an XML File with the SAX Parser

[Creating the Skeleton](#)

[Importing Classes](#)

[Setting up for I/O](#)

[Implementing the ContentHandler Interface](#)

[Setting up the Parser](#)

[Writing the Output](#)

[Spacing the Output](#)

[Handling Content Events](#)

[Compiling the Program](#)

[Running the Program](#)

[Command Scripts](#)

[Checking the Output](#)

[Identifying the Events](#)

[Compressing the Output](#)

[Inspecting the Output](#)

[Documents and Data](#)

2b. Adding Additional Event Handlers

[Identifying the Document's Location](#)

[Handling Processing Instructions](#)

[Summary](#)

3. Handling Errors with the Nonvalidating Parser

[Introducing an Error](#)

[Handling a SAXParseException](#)

[Handling a SAXException](#)

[Improving the SAXParseException Handler](#)

[Handling a ParserConfigurationException](#)

[Handling an IOException](#)

[Understanding NonFatal Errors](#)

[Handling NonFatal Errors](#)

[Handling Warnings](#)

[4. Substituting and Inserting Text](#)

[Handling Special Characters](#)

[Using an Entity Reference in an XML Document](#)

[Handling Text with XML-Style Syntax](#)

[Handling CDATA and Other Characters](#)

[5a. Creating a Document Type Definition \(DTD\)](#)

[Basic DTD Definitions](#)

[Defining Text and Nested Elements](#)

[Limitations of DTDs](#)

[Special Element Values in the DTD](#)

[Referencing the DTD](#)

[5b. DTD's Effect on the Nonvalidating Parser](#)

[Tracking Ignorable Whitespace](#)

[Cleanup](#)

[Documents and Data](#)

[Empty Elements, Revisited](#)

[5c. Defining Attributes and Entities in the DTD](#)

[Defining Attributes in the DTD](#)

[Defining Entities in the DTD](#)

[Echoing the Entity References](#)

[Additional Useful Entities](#)

[Referencing External Entities](#)

[Echoing the External Entity](#)

[Summarizing Entities](#)

[5d. Referencing Binary Entities](#)

[Using a MIME Data Type](#)

[The Alternative: Using Entity References](#)

[6. Using the Validating Parser](#)

[Configuring the Factory](#)

[Using the Environment Variable](#)

[Experimenting with Validation Errors](#)

[Error Handling in the Validating Parser](#)

[7a. Defining Parameter Entities and Conditional Sections](#)

[Creating and Referencing a Parameter Entity](#)

[Conditional Sections](#)

[7b. Parsing the Parameterized DTD](#)

[DTD Warnings](#)

[8. Handling Lexical Events](#)

[How the LexicalHandler Works](#)

[Working with a LexicalHandler](#)

[9. Using the DTDHandler and EntityResolver](#)

[The DTDHandler API](#)

[The EntityResolver API](#)

[Part III: XML and the Document Object Model \(DOM\)](#)

[1. Reading XML data into a DOM](#)

[Creating the Program](#)

[Additional Information](#)

[Looking Ahead](#)

[2a. Displaying a DOM Hierarchy](#)

[Echoing Tree Nodes](#)

[Convert DomEcho to a GUI App](#)

[Create Adapters to Display the DOM in a JTree](#)

[Finish it Up](#)

[2b. Examining the Structure of a DOM](#)

[Displaying a Simple Tree](#)

[Displaying a More Complex Tree](#)

[Finishing Up](#)

[3. Constructing a User-Friendly JTree from a DOM](#)

[Compressing the Tree View](#)

[Acting on Tree Selections](#)

[Handling Modifications](#)

[Finishing Up](#)

[4. Creating and Manipulating a DOM](#)

[Obtaining a DOM from the Factory](#)

[Normalizing the DOM](#)

[Other Operations](#)

[Finishing Up](#)

[5. Using Namespaces](#)

[Defining a Namespace](#)

[Referencing a Namespace](#)

[Defining a Namespace Prefix](#)

[Part IV: Using XSLT](#)

[1. Introducing XSLT and XPath](#)

[The XSLT Packages](#)

[How XPath Works](#)

[XPath Reference](#)

[2. Writing Out a DOM as an XML File](#)

[Reading the XML](#)

[Creating a Transformer](#)

[Writing the XML](#)

[Writing Out a Subtree of the DOM](#)

[Summary](#)

[3. Generating XML from an Arbitrary Data Structure](#)

[Creating a Simple File](#)

[Creating a Simple Parser](#)

[Modifying the "Parser" to Generate SAX Events](#)

[Using the Parser as a SAXSource](#)

[Doing the Conversion](#)

[4. Transforming XML Data with XSLT](#)

[Defining an Ultra-Simple `article` Document Type](#)

[Creating a Test Document](#)

[Writing an XSLT Transform](#)

[Processing the Basic Structure Elements](#)

[Writing the Basic Program](#)

[Trimming the Whitespace](#)

[Processing the Remaining Structure Elements](#)

[Process Inline \(Content\) Elements](#)

[Printing the HTML](#)

[What Else Can XSLT Do?](#)

[5. Concatenating XSLT Transformations with a Filter Chain](#)

[Writing the Program](#)

[Understanding How it Works](#)

[Testing the Program](#)

Additional Information

[Java's Encoding Schemes](#)



[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, `<`, 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 `<`; 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 (*). 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](#) `<` and `>` 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 `` and `` 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 [XML](#) specification that is concerned with identifying sections of documents so that they can be referenced in links or included in other documents.

XSL

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

- Specify an addressing mechanism, so you can identify the parts of an XML file that a transformation applies to. (**XPath**)
- Specify tag conversions, so you convert XML data into a different format. (**XSLT**)
- Specify display characteristics, such as 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 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

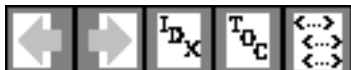
See [XSL](#).

Y

Z

_ (non-alpha)

[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](#)



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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

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.

Link Summary

Local Links

- [Manipulating Document Contents with the Document Object Model](#)

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.

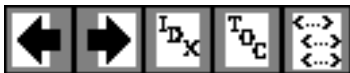
- **Converting Existing Data**

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:

1. [Writing a Simple XML File](#)
2. a) [Echoing an XML File with the SAX Parser](#)
b) [Adding Additional Event Handlers](#)
3. [Handling Errors with the Nonvalidating Parser](#)
4. [Substituting and Inserting Text](#)
5. a) [Creating a Document Type Definition \(DTD\)](#)
b) [DTD's Effect on the Nonvalidating Parser](#)
c) [Defining Attributes and Entities](#)
d) [Referencing Binary Entities](#)
6. [Using the Validating Parser](#)
7. a) [Defining Parameter Entities and Conditional Sections](#)
b) [Parsing the Parameterized DTD](#)
8. [Using a LexicalEventListener](#)
9. [Using the DTDHandler and EntityResolver](#)



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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

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.

Link Summary

Glossary Terms

[DOM](#), [element](#), [SAX](#)

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.

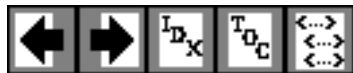
What You'll Learn

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:

1. [Reading XML data into a DOM](#)
2. a) [Displaying a DOM Hierarchy](#)
b) [Examining the Structure of a DOM](#)
3. [Constructing a User-Friendly JTree from a DOM](#)
4. [Creating and Manipulating a DOM](#)

5. [Using Namespaces](#)



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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. [Introducing 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*](#)



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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.

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:

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](#)

```
<!-- 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:

<i>Qualifier</i>	<i>Name</i>	<i>Meaning</i>
?	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 an error that says: `Illegal mixed content model for 'item'. Found (...`, where the hex character 28 is the angle bracket that 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>
```

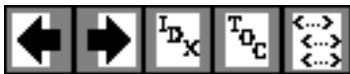
This tag defines the `slideshow` element as the [root](#) element for the document. An XML document must have exactly one root element. This is where that element is specified. In other words, this tag identifies the document *content* as a `slideshow`.

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](#)



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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.

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`

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](#)

specifies the element for which the attributes are being defined. In this case, the element is the `slide` 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.

<i>Attribute Type</i>	<i>Specifies...</i>
(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.

<i>Specification</i>	<i>Specifies...</i>
#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 `<` 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>
...
```

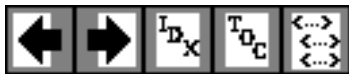
Note that the newline which follows the comment in the file is echoed as a character, but that the comment itself is ignored. That is the reason that the copyright message appears to start on the next line

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.



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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 `` 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

`<i>some text</i>` is well-formed, because the `<i>...</i>` sequence is completely nested within the `...` tag. This sequence, on the other hand, is not well-formed: `<i>some text</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](#)



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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

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 `&` 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.

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](#)

<i>Character</i>	<i>Reference</i>
&	&
<	<
>	>
"	"
'	'

Character References

A character reference like `“` 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 `<` 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 fizzle the frobmorten</item>
  <item>Then we framboze the staten</item>
  <item>Finally, we frenzle the fuznaten</item>
  <item><![CDATA[Diagram:

      frobmorten <----- fuznaten

```



```

      |               <3>               ^
      | <1>
      V
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>
      V
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](#).



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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 `<` and `&`. 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.

Note:

Lexical event handling is an 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.

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](#)

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](http://java.sun.com/xml/jaxp-1.1/docs/tutorial/sax/8_lex.html).

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
      |               ^
      |               |
      | <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](#)



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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

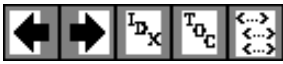
Now that you understand how certain instances of whitespace can be ignorable, it is time revise the definition of an "empty" element. That definition can now be expanded to include

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

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



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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:

- The [DTD](#) is required.
- Since the DTD is present, the `ignoreWhitespace` method is invoked whenever the DTD makes that possible.

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:

Link Summary

Exercise Links

- [Echo10.java](#)
- [Echo10-01](#)
- [Echo10-06](#)
- [Echo10-07](#)

API References

- [SAXParserFactory](#)

Glossary Terms

[DTD](#), [error](#), [XHTML](#)

```
> 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 `` 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 `` 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:

```

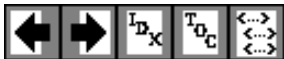
public void error(SAXParseException e)
throws SAXParseException
{
    throw e;
}

```

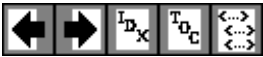
If that exception is not thrown, the validation errors are simply ignored.

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.



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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 `SAXExceptions` but not `IOExceptions`, 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 qualified 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 "&" or "<", 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.

	<i>UNIX</i>	<i>Windows</i>
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

```

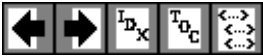
```
END_ELM: </em>
CHARS:   are great
END_ELM: </item>
```

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 `ignoreWhitespace` 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.



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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.

Creating and Referencing a Parameter Entity

Recall that the existing version of the slide presentation could not be validated because the document used `` 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](#).)

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](#)

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 `` and ``. 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.

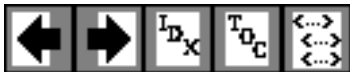
The solution is to use references to parameter entities in place of the `INCLUDE` and `IGNORE` keywords:

```
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.



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

2b. Adding Additional Event Handlers

Besides `ignoreWhitespace`, 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.

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](#).

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](#)

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.



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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.

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>
```

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](#)

```
...
```

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 validating 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='1.02' 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;
}

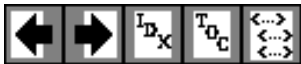
// dump warnings too
public void warning(SAXParseException err)
throws SAXParseException
{
    System.out.println("*** Warning"

```

```
        + ", 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.



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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](#).



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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).

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:

Link Summary

Local Links

- [Referencing Binary Entities](#)

API Links

- [DefaultHandler](#)
- [DTDHandler](#)
- [EntityResolver](#)
- [InputSource](#)

Glossary Terms

[notation](#), [SGML](#), [unparsed entity](#), [URL](#), [URN](#)

```

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 EntityResolver 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.



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

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.



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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.

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.

Link Summary

API Links

- [DTDHandler](#)

External Links

- [HTML 4.0 Specification](#)
- [MIME data types](#)

Glossary Terms

[namespace](#), [NOTATION](#), [SGML](#)

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*](#)



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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!

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);
        }
    }
}
```

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](#)


```

    }

    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 the 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](#)



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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 `<` 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 <code>PARA LIST</code> selects all <code>PARA</code> and <code>LIST</code> 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. <code>6 mod 4 = 2</code>)

Finally, expressions can be grouped in parentheses, so you don't have to worry about operator precedence. (Which, for those of you 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 bold word</PARA>`

the string-value of `<PARA>` is "This paragraph contains a bold word". In particular, note that `` 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 `"<"` 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 `<` or enclosed in a CDATA section. Similarly, occurrence of `"&"` will need to be converted to `&`.

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 occurrences 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.



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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.)

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:

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](#)


```
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](http://java.sun.com/xml/jaxp-1.1/docs/tutorial/dom/1_read.html) 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 {
        }

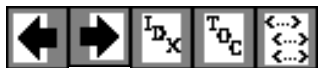
        // treat validation errors as fatal
        public void error(SAXParseException e)
            throws SAXParseException
        {
            throw e;
        }

        // 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.



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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.

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.

Link Summary

Exercise Links

- [DomEcho02.java](#)

External Links

- [DOM 2 Core Specification](#)
- [Understanding the TreeModel](#)

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](http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/), 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 be 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](http://java.sun.com/xml/jaxp-1.1/docs/tutorial/dom/2_display.html). That

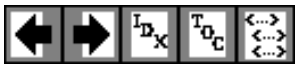
architecture was produced by Tom Santos and Steve Wilson, and is a lot more elegant than the quick hack going on here. It seemed worthwhile to put them here, though, so they would be immediately at hand when and if they're needed..

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.



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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.

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.

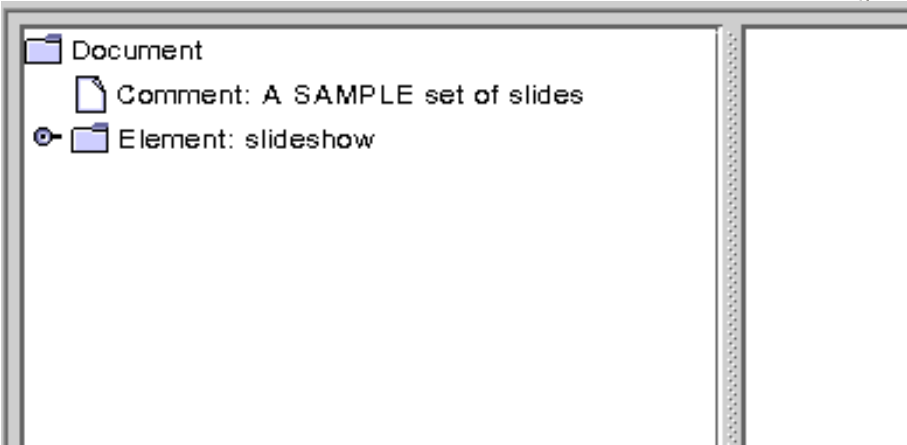


Figure 1: Document, Comment, and Element Nodes Displayed

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](#)

Glossary Terms

- [DTD](#)

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.

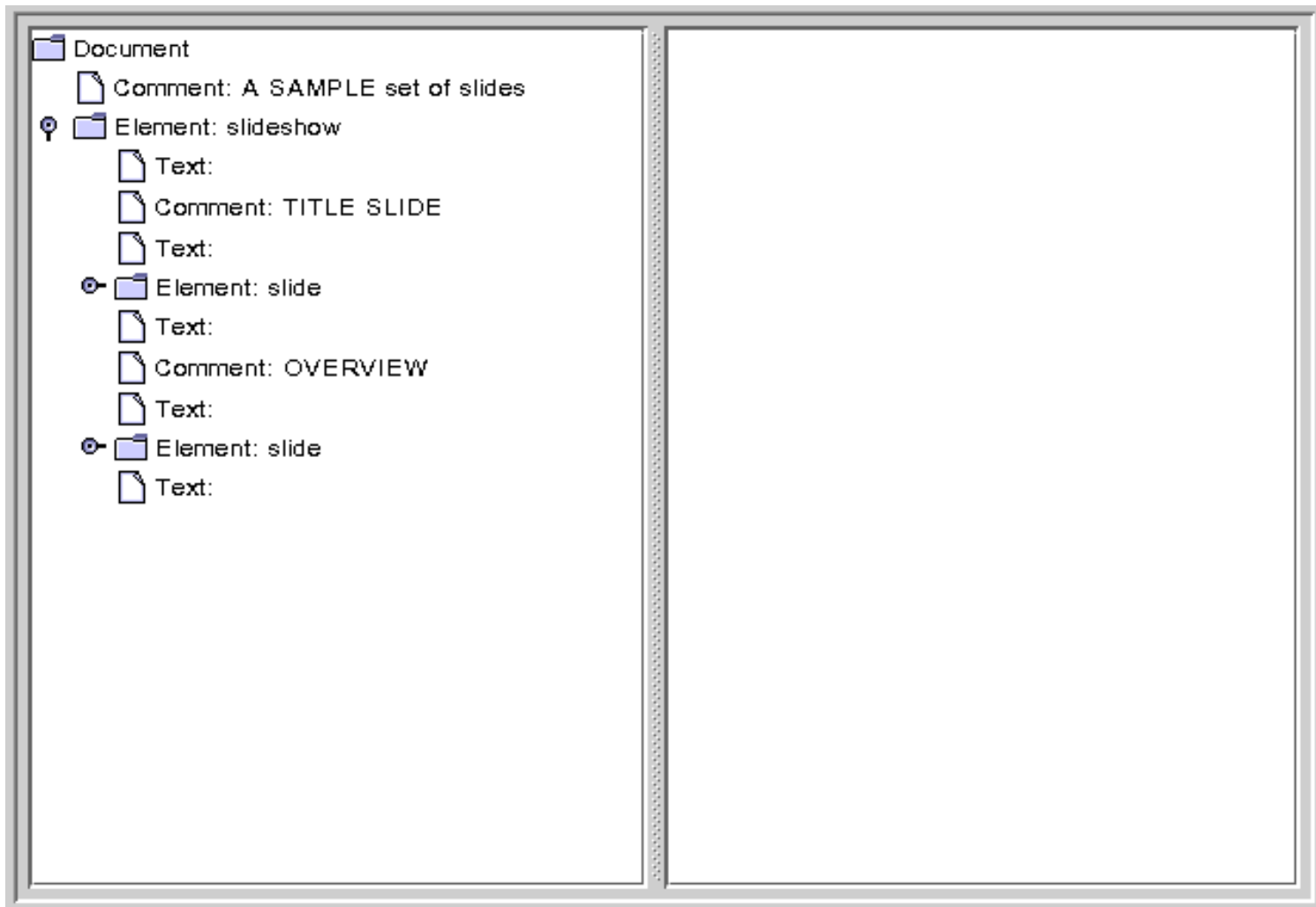


Figure 2: Element Node Expanded, No Attribute Nodes Showing

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 you 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](#).

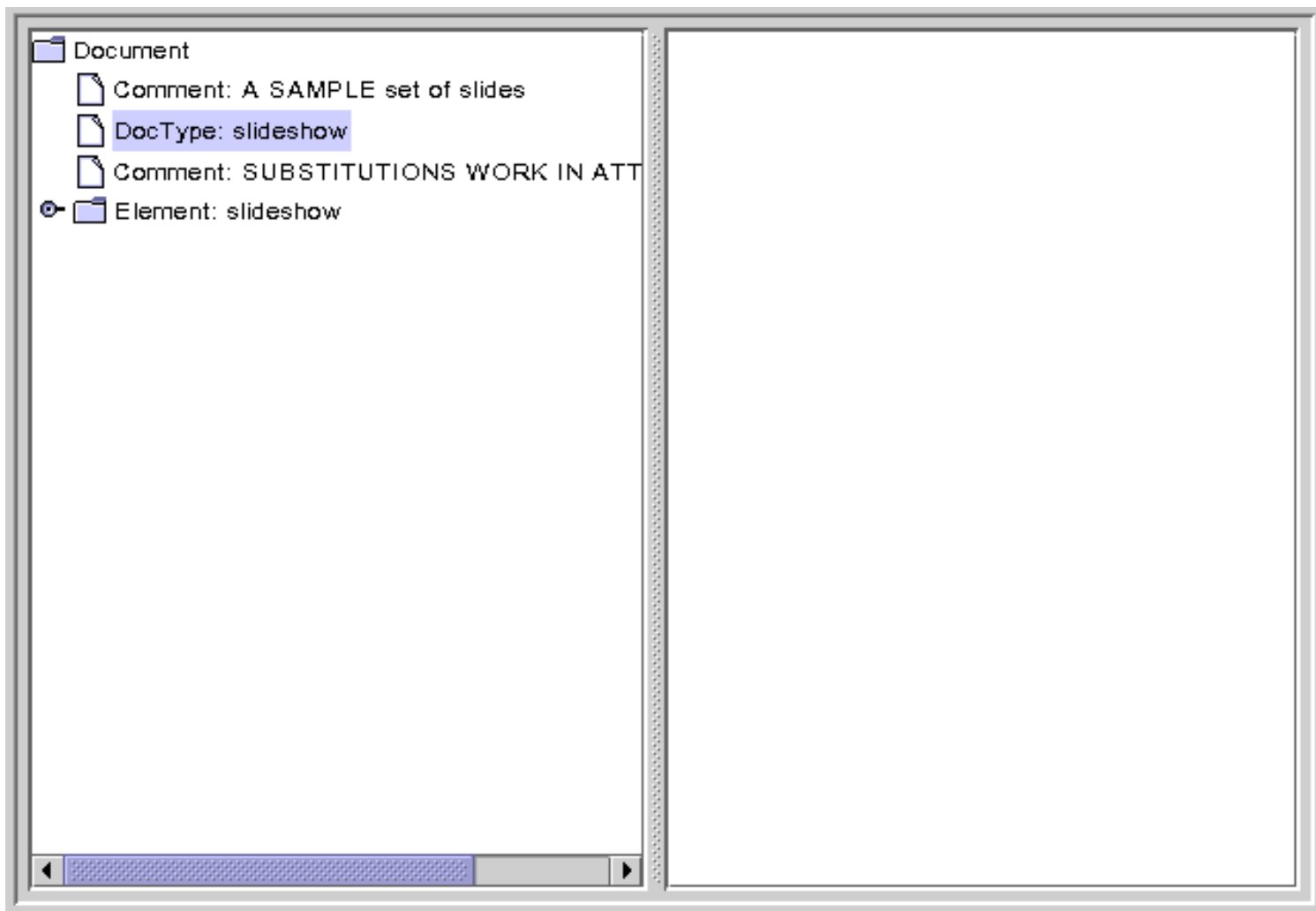


Figure 3: DocType Node Displayed

The DocType interface is actually an extension of `w3c.org.dom.Node`. It defines a `getEntities` method that you would use to 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.

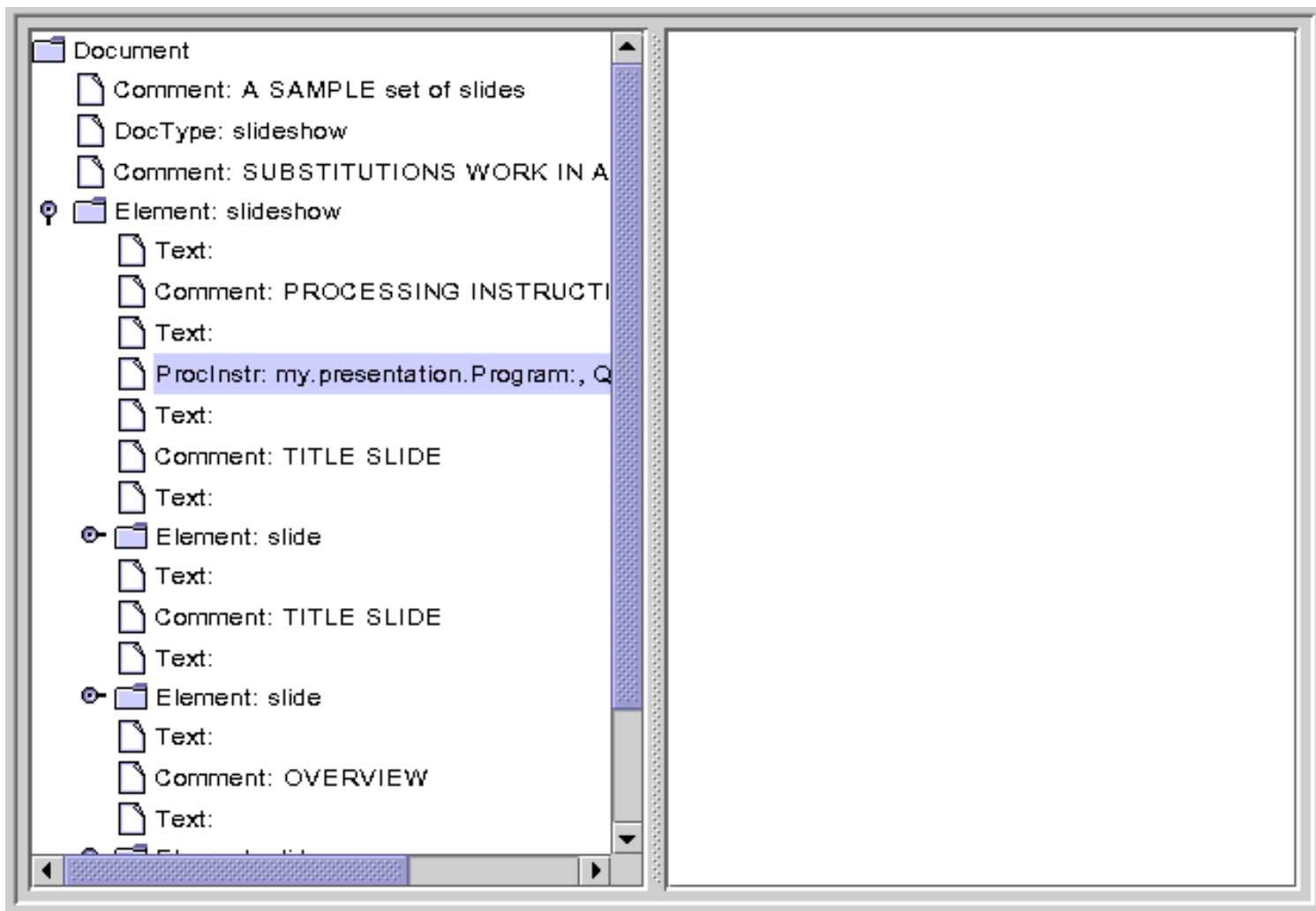


Figure 4: Processing Instruction Node Displayed

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 its 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.

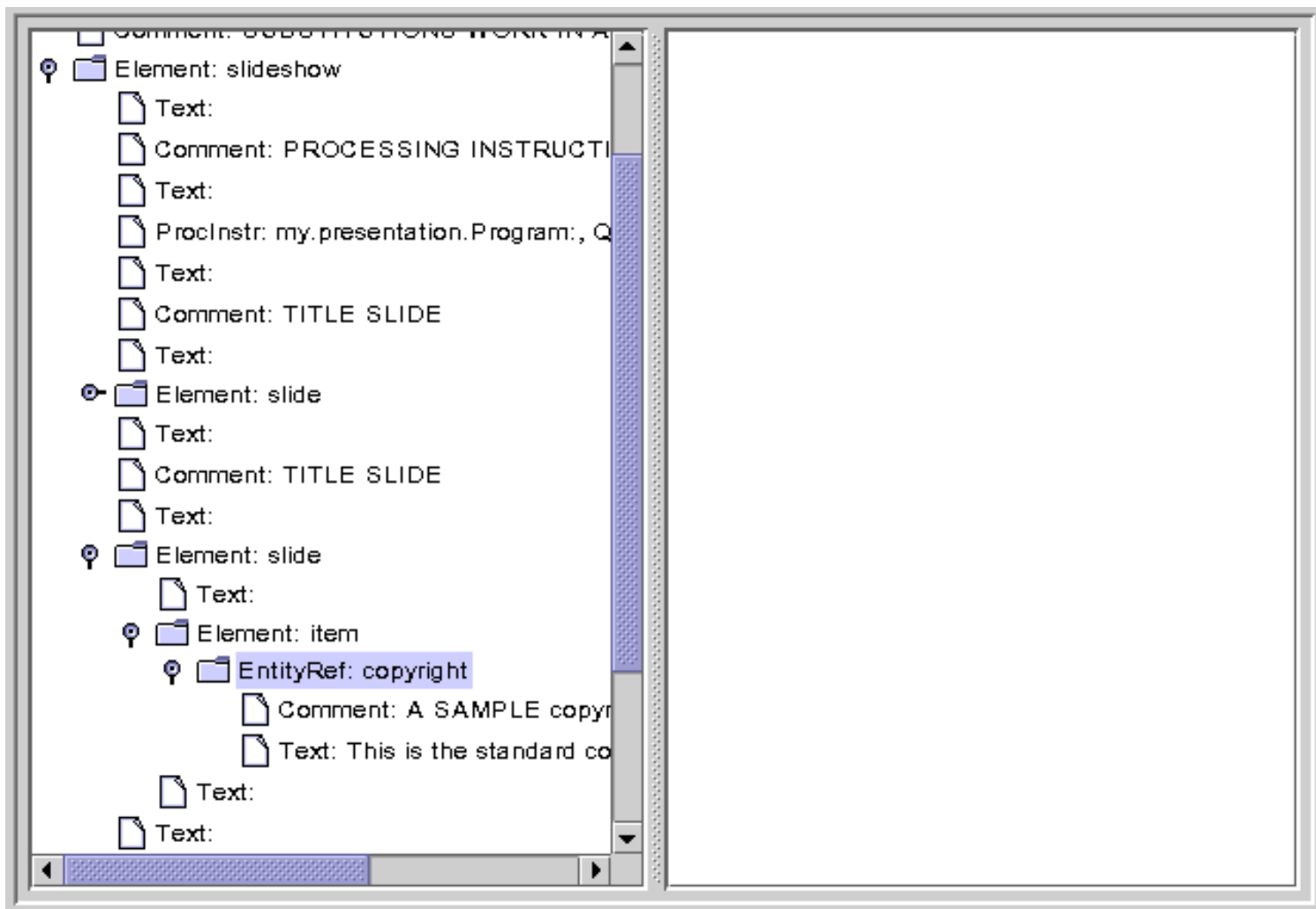


Figure 5: Entity Reference Node Displayed

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 conceivably 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.

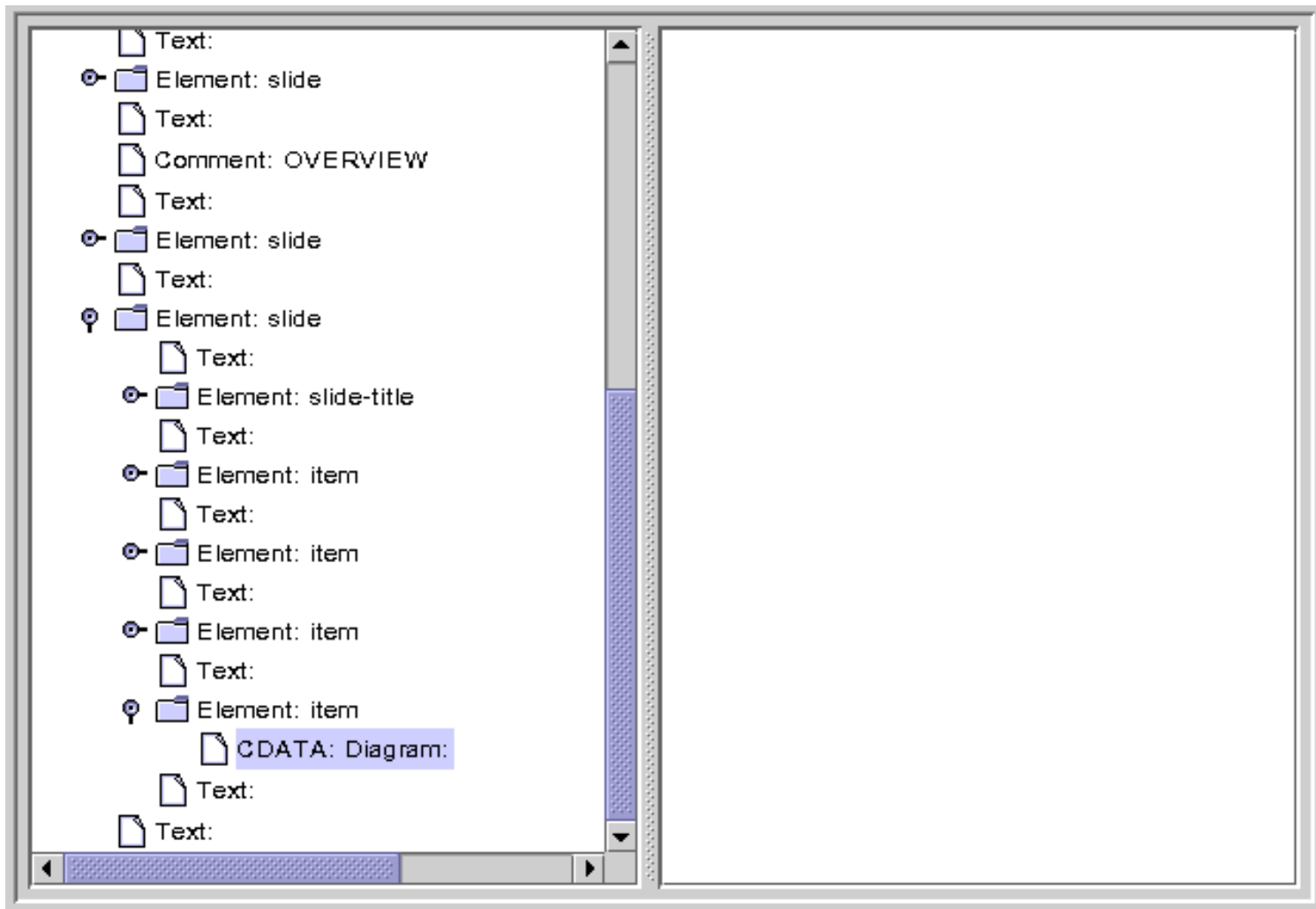
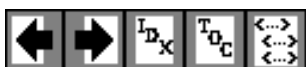


Figure 6: CDATA Node Displayed

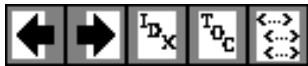
Here, the CDATA node is highlighted. Note that there are no nodes under it. Since a CDATA section is entirely uninterpreted, all of its contents are contained in the node's `value` property.

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.



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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.

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;
    ...
}
```

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](#)

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.

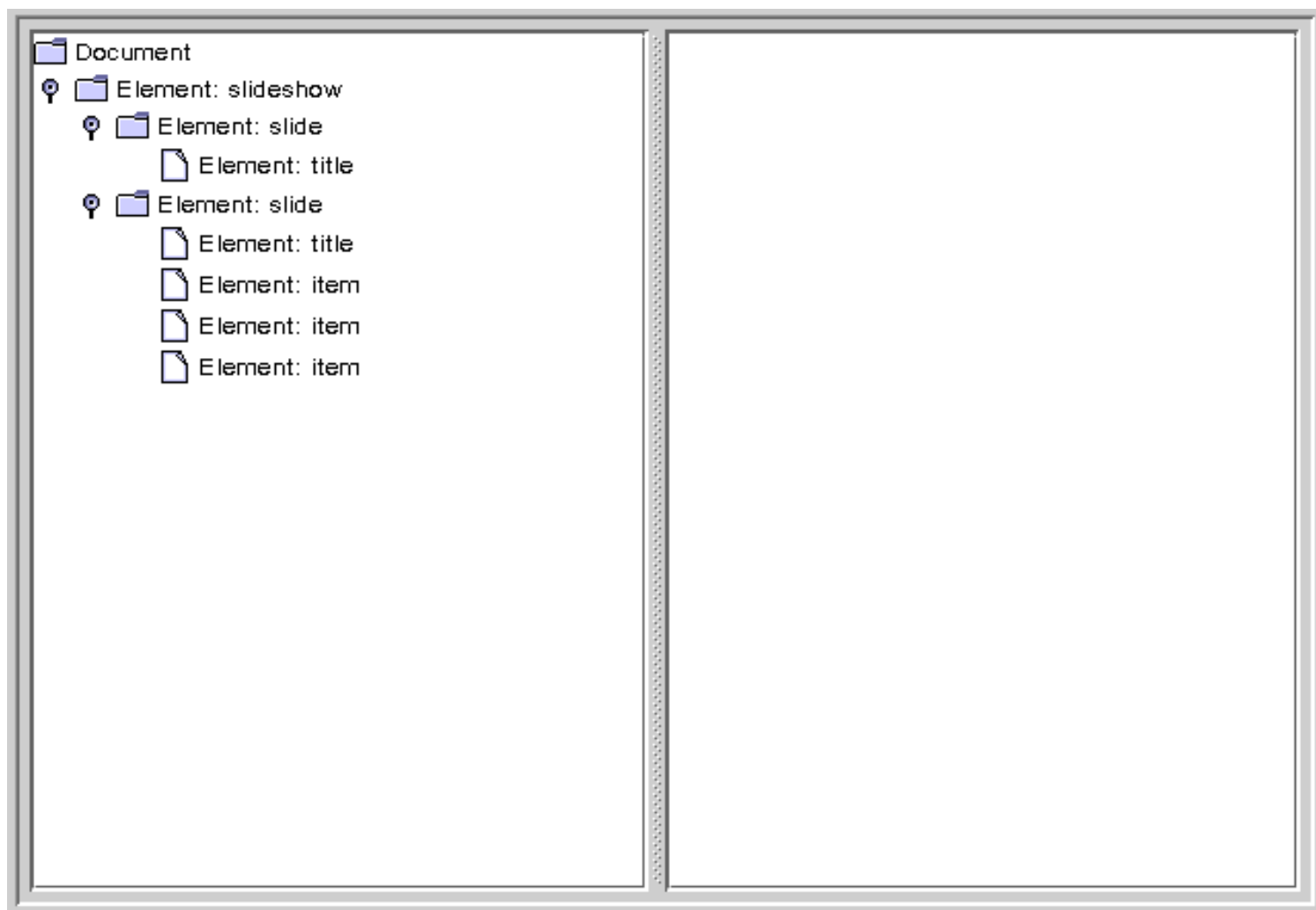


Figure 1: Tree View with a Collapsed Hierarchy

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 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 `` and `` 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.

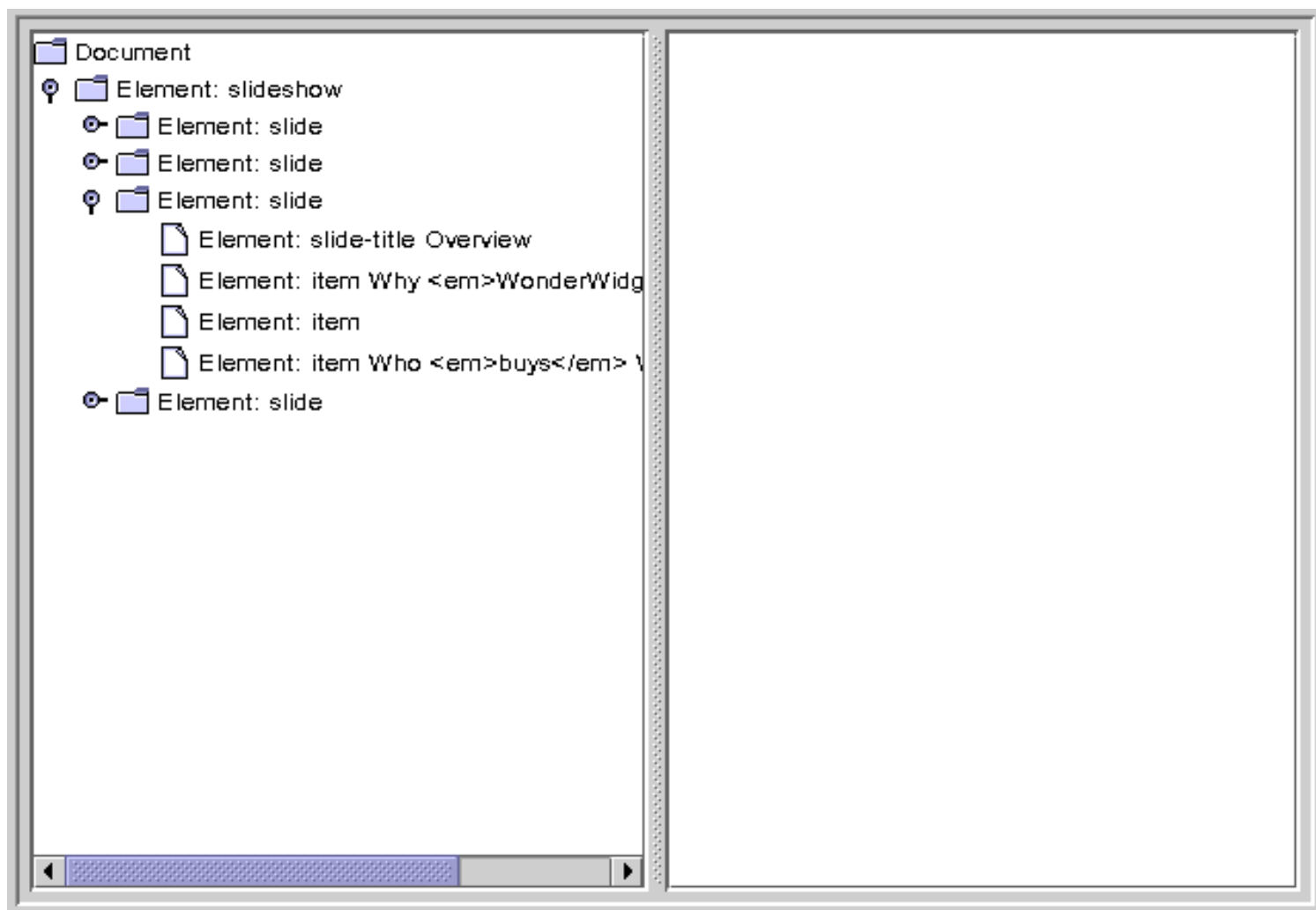


Figure 2: Collapsed Hierarchy Showing Text in Nodes

Selecting an item that includes XHTML subelements produces a display like that shown in Figure 3:

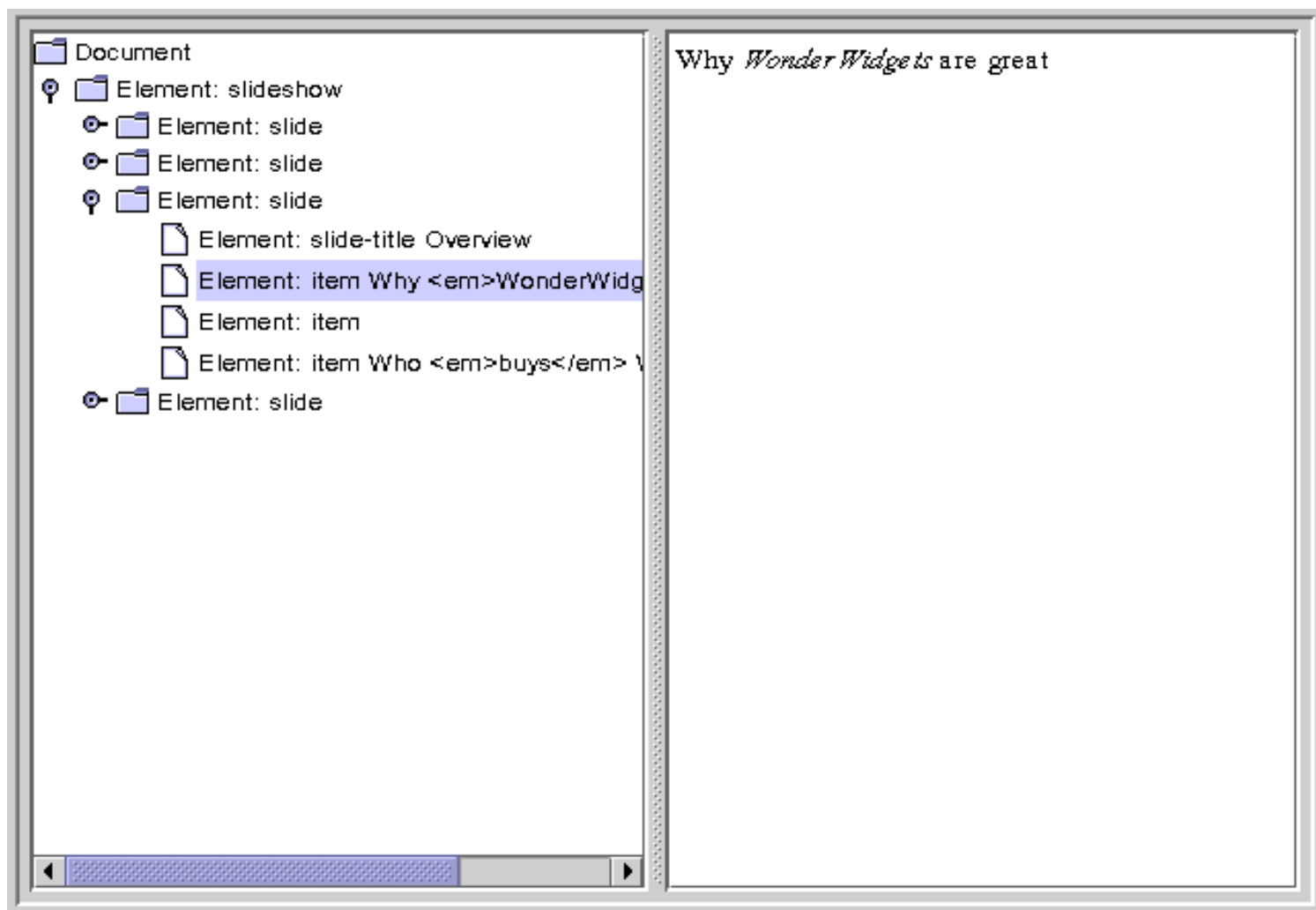


Figure 3: Node with ** Tag Selected

Selecting a node that contains an entity reference causes the entity text to be included, as shown in Figure 4:

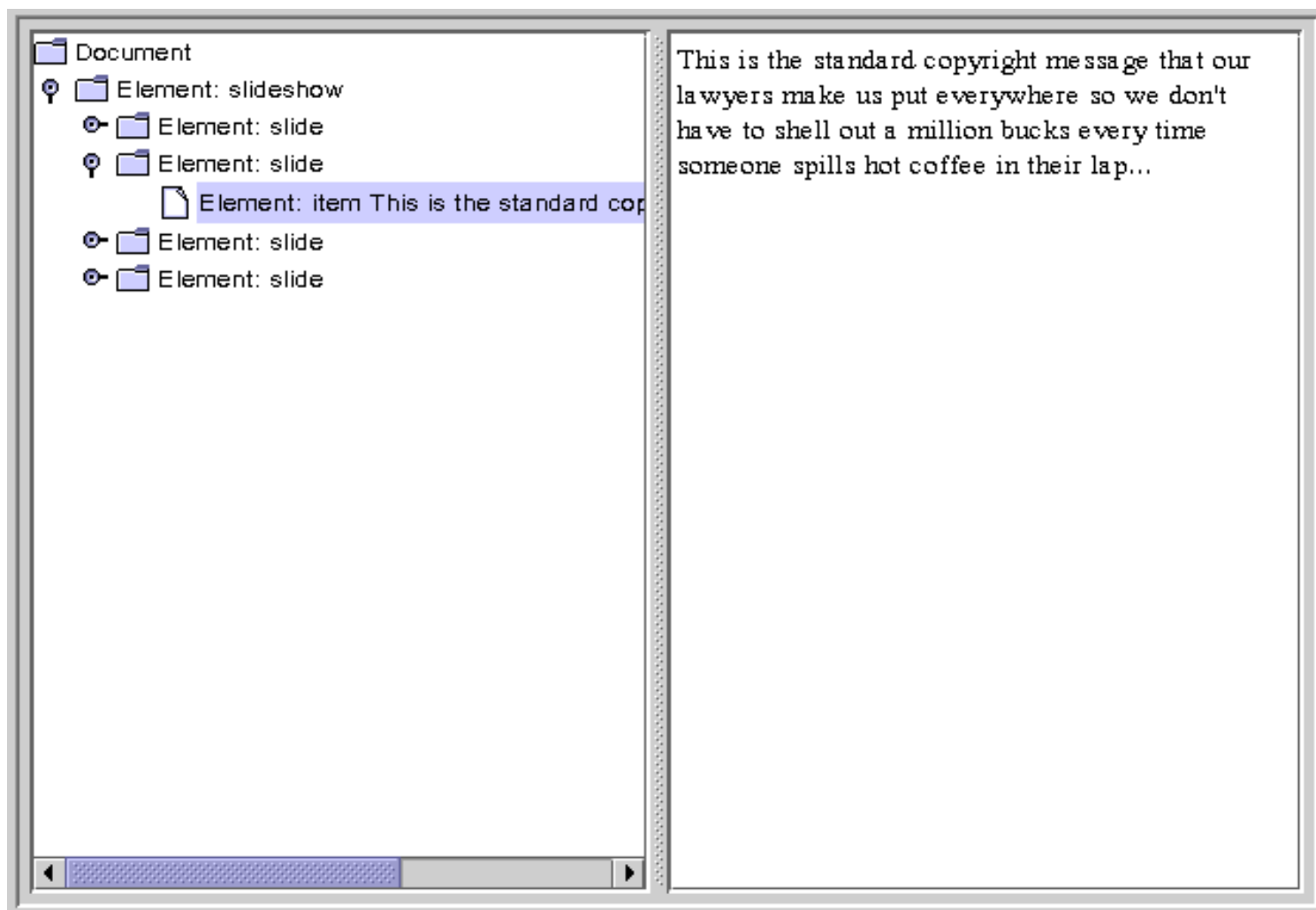


Figure 4: Node with Entity Reference Selected

Finally, selecting a node that includes a CDATA section produces results like those shown in Figure 5:

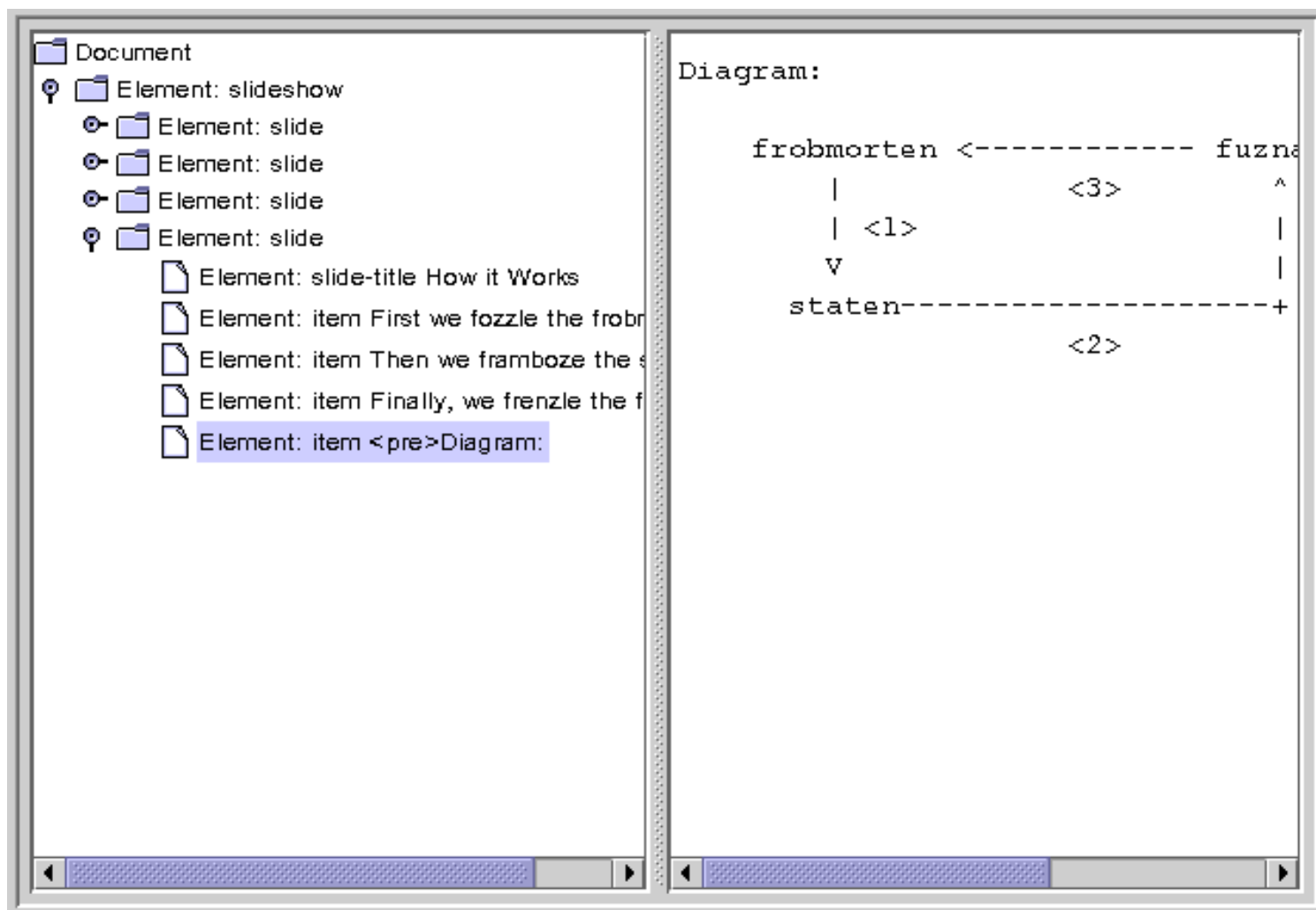


Figure 5: Node with CDATA Component Selected

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 ``, ``, `` 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](#)



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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.

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() {
        ...
    }
}
```

Link Summary

Exercise Links

- [DomEcho05.java](#)
- [DomEcho06.java](#)

API Links

- [org.w3c.dom.Node](#)
- [org.w3c.dom.Element](#)

Glossary Terms

[normalization](#)

```

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:

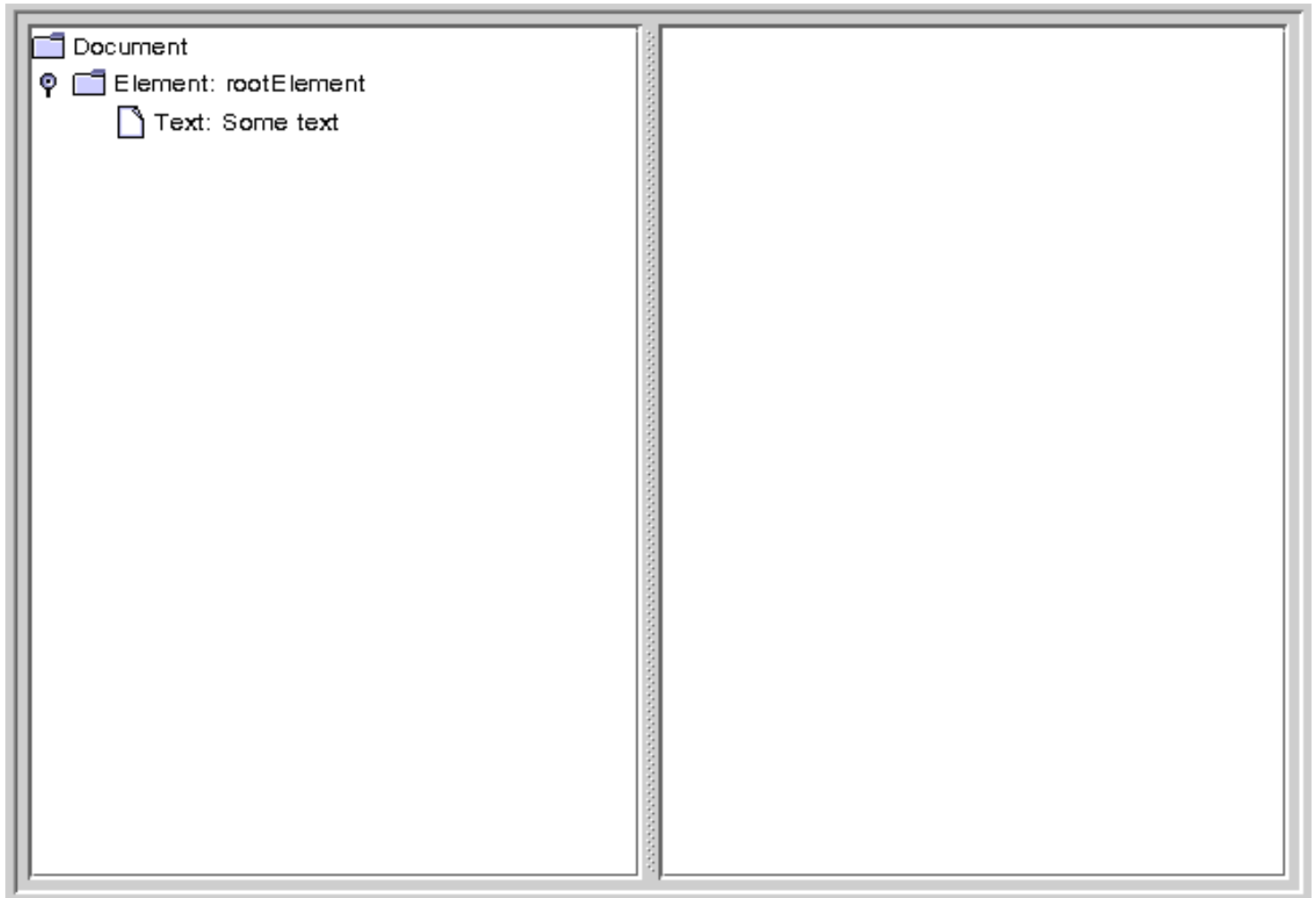


Figure 2: Text Nodes Merged After Normalization

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.)

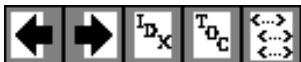
You can also use the Document's `createAttribute` operation to create an instance of `Attribute`, and use an overloaded version of `setAttribute` to add that.

Removing and Changing Nodes

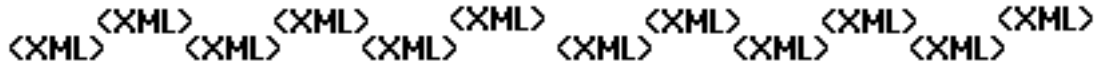
To remove a node, you use its parent [Node](#)'s `removeChild` method. To change it, you can either use the parent node's `replaceChild` operation or the node's `setNodeValue` operation.

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

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:

Link Summary

Local Links

- [Defining Attributes in the DTD](#)

External Links

- [Namespace Specification](#)

Glossary Terms

[attribute](#), [namespace](#), [URI](#), [URL](#), [URN](#)

```
<!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 its *default namespace*, as defined by name of the DTD file it resides in. As long as there 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 is 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:

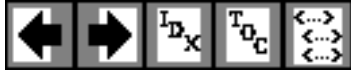
```
<SL:slideshow xmlns:SL='http://www.example.com/slideshow'
    ...>
    ...
    <slide>
        <SL:title>Overview<SL:title>
    </slide>
    ...
</SL:slideshow>
```

Finally, note that multiple prefixes can be defined in the same element, as shown here:

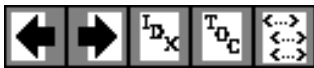
```
<SL:slideshow xmlns:SL='http://www.example.com/slideshow'
    xmlns:xhtml='urn:...'>
    ...
```

```
</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..



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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.

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:

Link Summary

Exercise Links

- [PersonalAddressBook.ldif](#)
- [AddressBookReader01.java](#)
- [AddressBookReaderLog01](#)
- [TransformationApp04.java](#)
- [TransformationLog04](#)

API Links

- [ContentHandler](#)
- [InputSource](#)
- [SAXSource](#)
- [XmlReader](#)

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
xmzillanickname: Fred
mail: Fred@barneys.house
xmzillausehtmlmail: TRUE
givenname: Fred
sn: Flinstone
telephonenumber: 999-Quarry
homephone: 999-BedrockLane
facsimiletelephonenumber: 888-Squawk
pagerphone: 777-pager
cellphone: 555-cell
xmzillaanyphone: 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 "xmzillausehtmlmail". The second argument gives the variable name stored in the file (xmzillausehtmlmail). 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 now created a transformer which will use a SAXSource as input, and produce a StreamResult as output..

Doing the Conversion

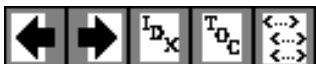
Now run the app on the address book file. Your output should look like this:

```

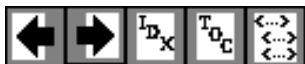
<?xml version="1.0" encoding="UTF-8"?>
<addressbook>
  <nickname>Fred</nickname>
  <email>fred@barneys.house</email>
  <html>TRUE</html>
  <firstname>Fred</firstname>
  <lastname>Flintstone</lastname>
  <work>999-Quarry</work>
  <home>999-BedrockLane</home>
  <fax>888-Squawk</fax>
  <pager>777-pager</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!



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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 generate 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.

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).

Link Summary

Local Links

- [XPath addressing](#)

Exercise Links

- [TransformationApp02](#)
- [Stylizer.java](#)
- [article1.xml](#) / [article1-xml.html](#)
- [article1a.xml](#) / [article1a-xsl.html](#)
- [stylizer1a.txt](#) / [stylizer1a.html](#)
- [article1b.xml](#) / [article1b-xsl.html](#)
- [stylizer1b.txt](#) / [stylizer1b.html](#)
- [article1c.xml](#) / [article1c-xsl.html](#)
- [stylizer1c.txt](#) / [stylizer1c.html](#)
- [article2.xml](#) / [article2-xml.html](#)
- [article2.xml](#) / [article2-xsl.html](#)
- [stylizer2.txt](#) / [stylizer2.html](#)
- [article3.xml](#) / [article3-xml.html](#)
- [article3.xml](#) / [article3-xsl.html](#)
- [stylizer3.txt](#) / [stylizer3.html](#)

API Links

- [Transformer](#)

External Links

- [Schematron](#) validator
- [XSLT Specification](#)
- [Java Printing Service \(JPS\)](#)

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:

- `` -- 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 be used for terms that are defined in the text. Such terms will be displayed in italics, the way they ordinarily are in a document. But using a special tag in the XML will allow an index 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 `http//...`. 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 `` 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 `` 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>..

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 TransformationAppStylizer
{
    if (argv.length != 1 2) {
        System.err.println ("Usage: java TransformationAppStylizer 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(argv[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 [stylyzer1b.txt](#). (The displayable versions are [article1b-xsl.html](#) and [stylyzer1b.html](#).)

To do remove some of the excess whitespace, add the line highlighted below to the stylesheet.

```

<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>

```

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.)

```

<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 stylyzer1c.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 [stylyzer2.txt](#). (The displayable versions are [article2-xml.html](#), [article2-xsl.html](#), and [stylyzer2.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 alphabetic, 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 `
` tag. To be well-formed XML, the tag must be specified in the stylesheet as `
`, but that tag is not recognized by many browsers. And while most browsers recognize the sequence `
</Br>`, they all treat it like a paragraph break, instead of a single line break.

In other words, the transformation *must* generate a `
` tag, but the stylesheet must specify `
`. 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 `
` to their HTML form, `
`, on output. That conversion is important, because most browsers do not recognize the empty-tags. Here is a list of the affected tags:

<ul style="list-style-type: none"> • area • base • basefont • Br • col 	<ul style="list-style-type: none"> • frame • hr • img • input 	<ul style="list-style-type: none"> • 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 `
` in the middle of it. The "`<xsl:output method='html' />`" solves the problem, so you can code `
` in the stylesheet, but get `
` 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 `<` entity reference, it will appear as the "<" character in the generated text. When XML is generated, on the other hand, the `<` entity reference in the stylesheet would be unchanged, so it would appear as `<` in the generated text.

Note:

If you actually want `<` to be generated as part of the HTML output, you'll need to encode it as `&lt;` -- that sequence becomes `<` on output, because only the `&` 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 now 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 ``, `<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="htmlLink">
  <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="htmlLink">
      <xsl:with-param name="dest" select="@target"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>

<xsl:template name="htmlLink">
  ...
```

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="htmlLink">
      <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.

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.

determined by where it is defined, the value of a global inline variable consists of text nodes, `` nodes, etc. that happen to exist at the root level. In other words, the value of such a variable, in this case, is null.

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)

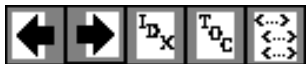
Loop over a collection of items and process each one, in turn.

- **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 alphabets (`format="A"`), lowercase alphabets (`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..



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



<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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.

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:

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](#)

```

    }
    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(stylesheets1));
XMLFilter filter2 = stf.newXMLFilter(new StreamSource(stylesheets2));

// 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 (...) {
    ...

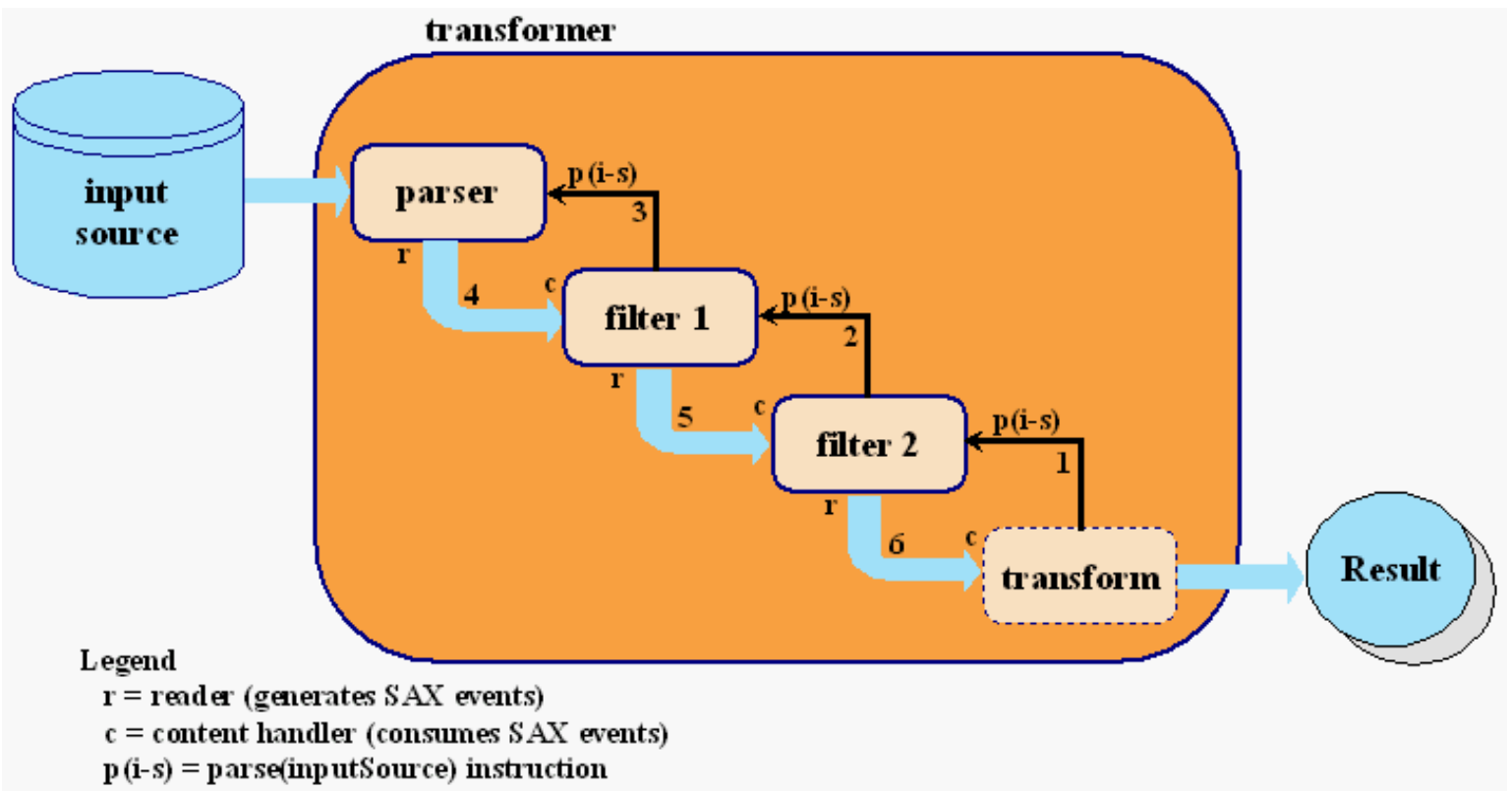
```

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 its "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.



When you create the transformer, you pass it 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 fizzle the frobmorten Then we framboze the staten Finally, we frenzle the fuznaten ^ | <1> | <1> = fizzle 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 fizzle 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:

CHARS: WonderWidgets

END_ELM:

CHARS: are great

END_ELM: </item>

ELEMENT: <item>

END_ELM: </item>

ELEMENT: <item>

CHARS: Who

ELEMENT:

CHARS: buys

END_ELM:

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:

CHARS: WonderWidgets

END_ELM:

CHARS: are great

END_ELM: </item>

ELEMENT: <item>

END_ELM: </item>

ELEMENT: <item>

CHARS: Who

ELEMENT:

CHARS: buys

```
        END_ELM: </em>
    CHARS:
    CHARS:    WonderWidgets
    END_ELM: </item>
END_ELM: </slide>
END_ELM: </slideshow>
END DOCUMENT
```

Work in Progress!!

<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

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>
```

Wake up to WonderWidgets!

Why *WonderWidgets* are great Who *buys* WonderWidgets

```
<?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:

CHARS: WonderWidgets

END_ELM:

CHARS: are great

END_ELM: </item>

CHARS:

ELEMENT: <item>

END_ELM: </item>

CHARS:

ELEMENT: <item>

CHARS: Who

ELEMENT:

CHARS: buys

END_ELM:

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 fizzle the frobmorten Then we framboze the staten Finally, we frenzle the fuznaten ^ | <1> | <1> = fizzle V | <2> = framboze staten-----+ <3> = frenzle <2>]]>

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:

CHARS: WonderWidgets

END_ELM:

CHARS: are great

END_ELM: </item>

CHARS:

ELEMENT: <item>

END_ELM: </item>

CHARS:

ELEMENT: <item>

CHARS: Who

ELEMENT:

CHARS: buys

END_ELM:

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
      |               ^
      | <1>           | <1> = fozzle
      v               | <2> = framboze
      staten-----+ <3> = frenzle
                  <2>
```

END_ELM: </item>

CHARS:

END_ELM: </slide>

CHARS:

END_ELM: </slideshow>

END DOCUMENT

```
/*
 * @(#)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 fizzle the frobmorten Then we framboze the staten Finally, we frenzle the fuznaten ^ |
<1> | <1> = fizzle 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 fizzle 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:
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
      |               ^
      | <1>           | <1> = fozzle
      v               | <2> = framboze
staten-----+ <3> = frenzle
              <2>
```

```
END_ELM: </item>
END_ELM: </slide>
END_ELM: </slideshow>
```

END DOCUMENT

```
/*
 * @(#)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;
static private String indentString = "    "; // Amount to indent
static 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("\n");
        }
    }
    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:

CHARS: WonderWidgets

END_ELM:

CHARS: are great

END_ELM: </item>

IGNORABLE

ELEMENT: <item>

END_ELM: </item>

IGNORABLE

ELEMENT: <item>

CHARS: Who

ELEMENT:

CHARS: buys

END_ELM:

CHARS: WonderWidgets


```
        END_ELM: </item>
    IGNOREABLE
    END_ELM: </slide>
IGNOREABLE
    ELEMENT: <slide
        ATTR: type    "exec"
    >
    IGNOREABLE
        ELEMENT: <title>
        CHARS:    Financial Forecast
        END_ELM: </title>
    IGNOREABLE
        ELEMENT: <item>
        CHARS:    Market Size
        CHARS:    <
        CHARS:    predicted!
        END_ELM: </item>
    IGNOREABLE
        ELEMENT: <item>
        CHARS:    Anticipated Penetration
        END_ELM: </item>
    IGNOREABLE
        ELEMENT: <item>
        CHARS:    Expected Revenues
        END_ELM: </item>
    IGNOREABLE
        ELEMENT: <item>
        CHARS:    Profit Margin
        END_ELM: </item>
    IGNOREABLE
    END_ELM: </slide>
IGNOREABLE
    ELEMENT: <slide
        ATTR: type    "tech"
    >
    IGNOREABLE
        ELEMENT: <title>
        CHARS:    How it Works
        END_ELM: </title>
    IGNOREABLE
        ELEMENT: <item>
        CHARS:    First we fozzle the frobmorten
        END_ELM: </item>
    IGNOREABLE
        ELEMENT: <item>
        CHARS:    Then we framboze the staten
        END_ELM: </item>
    IGNOREABLE
        ELEMENT: <item>
        CHARS:    Finally, we frenzle the fuznaten
        END_ELM: </item>
```

ELEMENT: `<item>`

frobmorten <----- fuznaten

 $| \langle 1 \rangle$

V

<2>

 \wedge

1

1

- +

<2> = framboze

<3> = frenzle

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("\n");
        }
    }
    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);
    }
}
```

}

```
/*
 * @(#)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%  %*%
```

```
/*
 * @(#)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>
```

```
/*
 * @(#)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\t");
            emit(attrs.getValue(i));
            emit("\n");
        }
    }
    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:

```
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("\n");
    }
}
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:

CHARS: WonderWidgets

END_ELM:

CHARS: are great

END_ELM: </item>

CHARS:

ELEMENT: <item>

END_ELM: </item>

CHARS:

ELEMENT: <item>

CHARS: Who

ELEMENT:

CHARS: buys

END_ELM:

CHARS: WonderWidgets

END_ELM: </item>

CHARS:

END_ELM: </slide>

CHARS :

END_ELM: </slideshow>

END DOCUMENT

Wake up to WonderWidgets!

Why *WonderWidgets* are great Who *buys* WonderWidgets


```
<?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:

CHARS: WonderWidgets

END_ELM:

CHARS: are great

END_ELM: </item>

CHARS:

ELEMENT: <item>

END_ELM: </item>

CHARS:

ELEMENT: <item>

CHARS: Who

ELEMENT:

CHARS: buys

END_ELM:

CHARS: WonderWidgets

END_ELM: </item>

CHARS:

END_ELM: </slide>

CHARS:

END_ELM: </slideshow>

END DOCUMENT

Wake up to WonderWidgets!

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:

CHARS: WonderWidgets

END_ELM:

CHARS: are great

END_ELM: </item>

CHARS:

ELEMENT: <item>

CHARS:

ELEMENT: <item>

CHARS: Who

ELEMENT:

CHARS: buys

END_ELM:

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("\n");
        }
    }
    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:

CHARS: WonderWidgets

END_ELM:

CHARS: are great

END_ELM: </item>

CHARS:

ELEMENT: <item>

CHARS:

ELEMENT: <item>

CHARS: Who

ELEMENT:

CHARS: buys

END_ELM:

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)
```

Wake up to WonderWidgets!

Why *WonderWidgets* are great Who *buys* WonderWidgets

```
<?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:

CHARS: WonderWidgets

END_ELM:

CHARS: are great

END_ELM: </item>

CHARS:

ELEMENT: <item>

END_ELM: </item>

CHARS:

ELEMENT: <item>

CHARS: Who

ELEMENT:

```
        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;
static private String indentString = "    "; // Amount to indent
static 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("\n");
        }
    }
    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 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:

CHARS: WonderWidgets

END_ELM:

CHARS: are great

END_ELM: </item>

ELEMENT: <item>

END_ELM: </item>

ELEMENT: <item>

CHARS: Who

ELEMENT:

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 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="htmlLink">
      <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="htmlLink">
      <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="htmlLink">
  <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.

```
/*
 * @(#)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 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

<XML> <XML> <XML> <XML> <XML> <XML> <XML> <XML>

[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.

[All Classes](#)

Packages

[javax.xml.parsers](#)
[javax.xml.transform](#)
[javax.xml.transform.dom](#)
[javax.xml.transform.sax](#)
[javax.xml.transform.stream](#)
[org.w3c.dom](#)
[org.xml.sax](#)
[org.xml.sax.ext](#)
[org.xml.sax.helpers](#)

All Classes

[Attr](#)
[AttributeList](#)
[AttributeListImpl](#)
[Attributes](#)
[AttributesImpl](#)
[CDATASection](#)
[CharacterData](#)
[Comment](#)
[ContentHandler](#)
[DeclHandler](#)
[DefaultHandler](#)
[Document](#)
[DocumentBuilder](#)
[DocumentBuilderFactory](#)
[DocumentFragment](#)
[DocumentHandler](#)
[DocumentType](#)
[DOMException](#)
[DOMImplementation](#)
[DOMLocator](#)
[DOMResult](#)
[DOMSource](#)
[DTDHandler](#)
[Element](#)
[Entity](#)
[EntityReference](#)
[EntityResolver](#)
[ErrorHandler](#)
[ErrorListener](#)
[FactoryConfigurationError](#)
[HandlerBase](#)
[InputSource](#)

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)

[PREV](#)
[NEXT](#)

[FRAMES](#)
[NO FRAMES](#)

Java API for XML Parsing 1.1.1

Packages

javax.xml.parsers	Provides classes allowing the processing of XML documents.
javax.xml.transform	This package defines the generic APIs for processing transformation instructions, and performing a transformation from source to result.
javax.xml.transform.dom	This package implements DOM-specific transformation APIs. The DOMSource class allows the client of the implementation of this API to specify a DOM Node as the source of the input tree.
javax.xml.transform.sax	This package implements SAX2-specific transformation APIs.
javax.xml.transform.stream	This package implements stream- and URI-specific transformation APIs.
org.w3c.dom	
org.xml.sax	
org.xml.sax.ext	
org.xml.sax.helpers	

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)

[PREV](#)
[NEXT](#)

[FRAMES](#)
[NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[LexicalHandler](#)
[Locator](#)
[LocatorImpl](#)
[NamedNodeMap](#)
[NamespaceSupport](#)
[Node](#)
[NodeList](#)
[Notation](#)
[OutputKeys](#)
[Parser](#)
[ParserAdapter](#)
[ParserConfigurationException](#)
[ParserFactory](#)
[ProcessingInstruction](#)
[Result](#)
[SAXException](#)
[SAXNotRecognizedException](#)
[SAXNotSupportedException](#)
[SAXParseException](#)
[SAXParser](#)
[SAXParserFactory](#)
[SAXResult](#)
[SAXSource](#)
[SAXTransformerFactory](#)
[Source](#)
[SourceLocator](#)
[StreamResult](#)
[StreamSource](#)
[Templates](#)
[TemplatesHandler](#)
[Text](#)
[Transformer](#)
[TransformerConfigurationException](#)
[TransformerException](#)
[TransformerFactory](#)
[TransformerFactoryConfigurationError](#)
[TransformerHandler](#)
[URIResolver](#)
[XMLFilter](#)
[XMLFilterImpl](#)
[XMLReader](#)
[XMLReaderAdapter](#)
[XMLReaderFactory](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform.dom

Class DOMSource

java.lang.Object

|
+-- **javax.xml.transform.dom.DOMSource**public class **DOMSource**

extends java.lang.Object

implements [Source](#)

Acts as a holder for a transformation Source tree in the form of a Document Object Model (DOM) tree.

See Also:[Document Object Model \(DOM\) Level 2 Specification](#)

Field Summary

static java.lang.String	FEATURE If TransformerFactory.getFeature(java.lang.String) returns true when passed this value as an argument, the Transformer supports Source input of this type.
-------------------------	--

Constructor Summary

DOMSource () Zero-argument default constructor.
DOMSource (Node n) Create a new input source with a DOM node.
DOMSource (Node node, java.lang.String systemID) Create a new input source with a DOM node, and with the system ID also passed in as the base URI.

Method Summary

Node	getNode () Get the node that represents a Source DOM tree.
java.lang.String	getSystemId () Get the base ID (URL or system ID) from where URLs will be resolved.
void	setNode (Node node) Set the node that will represents a Source DOM tree.
void	setSystemId (java.lang.String baseID) Set the base ID (URL or system ID) from where URLs will be resolved.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

FEATURE

public static final java.lang.String **FEATURE**

If [TransformerFactory.getFeature\(java.lang.String\)](#) returns true when passed this value as an argument, the Transformer supports Source input of this type.

Constructor Detail

DOMSource

public **DOMSource**()

Zero-argument default constructor. If this is used, and no DOM source is set, then the Transformer will create an empty source Document using [DocumentBuilder.newDocument\(\)](#).

DOMSource

public **DOMSource**([Node](#) n)

Create a new input source with a DOM node. The operation will be applied to the subtree rooted at

this node. In XSLT, a "/" pattern still means the root of the tree (not the subtree), and the evaluation of global variables and parameters is done from the root node also.

Parameters:

n - The DOM node that will contain the Source tree.

DOMSource

```
public DOMSource(Node node,  
                 java.lang.String systemID)
```

Create a new input source with a DOM node, and with the system ID also passed in as the base URI.

Parameters:

node - The DOM node that will contain the Source tree.

systemID - Specifies the base URI associated with node.

Method Detail

setNode

```
public void setNode(Node node)
```

Set the node that will represents a Source DOM tree.

Parameters:

node - The node that is to be transformed.

getNode

```
public Node getNode()
```

Get the node that represents a Source DOM tree.

Returns:

The node that is to be transformed.

setSystemId

```
public void setSystemId(java.lang.String baseID)
```

Set the base ID (URL or system ID) from where URLs will be resolved.

Specified by:[setSystemId](#) in interface [Source](#)**Parameters:**

baseID - Base URL for this DOM tree.

getSystemId

```
public java.lang.String getSystemId()
```

Get the base ID (URL or system ID) from where URLs will be resolved.

Specified by:[getSystemId](#) in interface [Source](#)**Returns:**

Base URL for this DOM tree.

Overview	Package	Class	Use	Tree	Deprecated	Index	Help
--------------------------	-------------------------	--------------	---------------------	----------------------	----------------------------	-----------------------	----------------------

[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.comCopyright 1998-2001 [Sun Microsystems Inc.](#)All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

Hierarchy For All Packages

Package Hierarchies:

[javax.xml.parsers](#), [javax.xml.transform](#), [javax.xml.transform.dom](#), [javax.xml.transform.sax](#),
[javax.xml.transform.stream](#), [org.w3c.dom](#), [org.xml.sax](#), [org.xml.sax.ext](#), [org.xml.sax.helpers](#)

Class Hierarchy

- class java.lang.Object
 - class org.xml.sax.helpers.[AttributeListImpl](#) (implements org.xml.sax.[AttributeList](#))
 - class org.xml.sax.helpers.[AttributesImpl](#) (implements org.xml.sax.[Attributes](#))
 - class org.xml.sax.helpers.[DefaultHandler](#) (implements org.xml.sax.[ContentHandler](#), org.xml.sax.[DTDHandler](#), org.xml.sax.[EntityResolver](#), org.xml.sax.[ErrorHandler](#))
 - class javax.xml.parsers.[DocumentBuilder](#)
 - class javax.xml.parsers.[DocumentBuilderFactory](#)
 - class javax.xml.transform.dom.[DOMResult](#) (implements javax.xml.transform.[Result](#))
 - class javax.xml.transform.dom.[DOMSource](#) (implements javax.xml.transform.[Source](#))
 - class org.xml.sax.[HandlerBase](#) (implements org.xml.sax.[DocumentHandler](#), org.xml.sax.[DTDHandler](#), org.xml.sax.[EntityResolver](#), org.xml.sax.[ErrorHandler](#))
 - class org.xml.sax.[InputSource](#)
 - class org.xml.sax.helpers.[LocatorImpl](#) (implements org.xml.sax.[Locator](#))
 - class org.xml.sax.helpers.[NamespaceSupport](#)
 - class javax.xml.transform.[OutputKeys](#)
 - class org.xml.sax.helpers.[ParserAdapter](#) (implements org.xml.sax.[DocumentHandler](#), org.xml.sax.[XMLReader](#))
 - class org.xml.sax.helpers.[ParserFactory](#)
 - class javax.xml.parsers.[SAXParser](#)
 - class javax.xml.parsers.[SAXParserFactory](#)
 - class javax.xml.transform.sax.[SAXResult](#) (implements javax.xml.transform.[Result](#))
 - class javax.xml.transform.sax.[SAXSource](#) (implements javax.xml.transform.[Source](#))
 - class javax.xml.transform.stream.[StreamResult](#) (implements javax.xml.transform.[Result](#))

- class javax.xml.transform.stream.[StreamSource](#) (implements javax.xml.transform.[Source](#))
- class java.lang.Throwable (implements java.io.Serializable)
 - class java.lang.Error
 - class javax.xml.parsers.[FactoryConfigurationError](#)
 - class javax.xml.transform.[TransformerFactoryConfigurationError](#)
- class java.lang.Exception
 - class javax.xml.parsers.[ParserConfigurationException](#)
 - class java.lang.RuntimeException
 - class org.w3c.dom.[DOMException](#)
 - class org.xml.sax.[SAXException](#)
 - class org.xml.sax.[SAXNotRecognizedException](#)
 - class org.xml.sax.[SAXNotSupportedException](#)
 - class org.xml.sax.[SAXParseException](#)
 - class javax.xml.transform.[TransformerException](#)
 - class javax.xml.transform.[TransformerConfigurationException](#)
- class javax.xml.transform.[Transformer](#)
- class javax.xml.transform.[TransformerFactory](#)
 - class javax.xml.transform.sax.[SAXTransformerFactory](#)
- class org.xml.sax.helpers.[XMLFilterImpl](#) (implements org.xml.sax.[ContentHandler](#), org.xml.sax.[DTDHandler](#), org.xml.sax.[EntityResolver](#), org.xml.sax.[ErrorHandler](#), org.xml.sax.[XMLFilter](#))
- class org.xml.sax.helpers.[XMLReaderAdapter](#) (implements org.xml.sax.[ContentHandler](#), org.xml.sax.[Parser](#))
- class org.xml.sax.helpers.[XMLReaderFactory](#)

Interface Hierarchy

- interface org.xml.sax.[AttributeList](#)
- interface org.xml.sax.[Attributes](#)
- interface org.xml.sax.[ContentHandler](#)
 - interface javax.xml.transform.sax.[TemplatesHandler](#)
 - interface javax.xml.transform.sax.[TransformerHandler](#) (also extends org.xml.sax.[DTDHandler](#), org.xml.sax.ext.[LexicalHandler](#))
- interface org.xml.sax.ext.[DeclHandler](#)
- interface org.xml.sax.[DocumentHandler](#)

- interface org.w3c.dom.[DOMImplementation](#)
- interface org.xml.sax.[DTDHandler](#)
 - interface javax.xml.transform.sax.[TransformerHandler](#)(also extends org.xml.sax.[ContentHandler](#), org.xml.sax.ext.[LexicalHandler](#))
- interface org.xml.sax.[EntityResolver](#)
- interface org.xml.sax.[ErrorHandler](#)
- interface javax.xml.transform.[ErrorListener](#)
- interface org.xml.sax.ext.[LexicalHandler](#)
 - interface javax.xml.transform.sax.[TransformerHandler](#)(also extends org.xml.sax.[ContentHandler](#), org.xml.sax.[DTDHandler](#))
- interface org.xml.sax.[Locator](#)
- interface org.w3c.dom.[NamedNodeMap](#)
- interface org.w3c.dom.[Node](#)
 - interface org.w3c.dom.[Attr](#)
 - interface org.w3c.dom.[CharacterData](#)
 - interface org.w3c.dom.[Comment](#)
 - interface org.w3c.dom.[Text](#)
 - interface org.w3c.dom.[CDATASection](#)
 - interface org.w3c.dom.[Document](#)
 - interface org.w3c.dom.[DocumentFragment](#)
 - interface org.w3c.dom.[DocumentType](#)
 - interface org.w3c.dom.[Element](#)
 - interface org.w3c.dom.[Entity](#)
 - interface org.w3c.dom.[EntityReference](#)
 - interface org.w3c.dom.[Notation](#)
 - interface org.w3c.dom.[ProcessingInstruction](#)
- interface org.w3c.dom.[NodeList](#)
- interface org.xml.sax.[Parser](#)
- interface javax.xml.transform.[Result](#)
- interface javax.xml.transform.[Source](#)
- interface javax.xml.transform.[SourceLocator](#)
 - interface javax.xml.transform.dom.[DOMLocator](#)
- interface javax.xml.transform.[Templates](#)
- interface javax.xml.transform.[URIResolver](#)

- interface org.xml.sax.[XMLReader](#)
- interface org.xml.sax.[XMLFilter](#)

[Overview](#) [Package](#) [Class](#) [Use](#) **[Tree](#)** [Deprecated](#) [Index](#) [Help](#)

[PREV](#) [NEXT](#)

[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) **Deprecated** [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Deprecated API

Deprecated Classes

[org.xml.sax.helpers.AttributeListImpl](#)

This class implements a deprecated interface, [AttributeList](#); that interface has been replaced by [Attributes](#), which is implemented in the [AttributesImpl](#) helper class.

[org.xml.sax.HandlerBase](#)

This class works with the deprecated [DocumentHandler](#) interface. It has been replaced by the SAX2 [DefaultHandler](#) class.

[org.xml.sax.helpers.ParserFactory](#)

This class works with the deprecated [Parser](#) interface.

Deprecated Interfaces

[org.xml.sax.AttributeList](#)

This interface has been replaced by the SAX2 [Attributes](#) interface, which includes Namespace support.

[org.xml.sax.DocumentHandler](#)

This interface has been replaced by the SAX2 [ContentHandler](#) interface, which includes Namespace support.

[org.xml.sax.Parser](#)

This interface has been replaced by the SAX2 [XMLReader](#) interface, which includes Namespace support.

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) **Deprecated** [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

How This API Document Is Organized

This API (Application Programming Interface) document has pages corresponding to the items in the navigation bar, described as follows.

Overview

The [Overview](#) page is the front page of this API document and provides a list of all packages with a summary for each. This page can also contain an overall description of the set of packages.

Package

Each package has a page that contains a list of its classes and interfaces, with a summary for each. This page can contain four categories:

- Interfaces (*italic*)
- Classes
- Exceptions
- Errors

Class/Interface

Each class, interface, inner class and inner interface has its own separate page. Each of these pages has three sections consisting of a class/interface description, summary tables, and detailed member descriptions:

- Class inheritance diagram
- Direct Subclasses
- All Known Subinterfaces
- All Known Implementing Classes
- Class/interface declaration
- Class/interface description
- Inner Class Summary
- Field Summary
- Constructor Summary
- Method Summary
- Field Detail

- [Constructor Detail](#)
- [Method Detail](#)

Each summary entry contains the first sentence from the detailed description for that item. The summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer.

Use

Each documented package, class and interface has its own Use page. This page describes what packages, classes, methods, constructors and fields use any part of the given class or package. Given a class or interface A, its Use page includes subclasses of A, fields declared as A, methods that return A, and methods and constructors with parameters of type A. You can access this page by first going to the package, class or interface, then clicking on the "Use" link in the navigation bar.

Tree (Class Hierarchy)

There is a [Class Hierarchy](#) page for all packages, plus a hierarchy for each package. Each hierarchy page contains a list of classes and a list of interfaces. The classes are organized by inheritance structure starting with `java.lang.Object`. The interfaces do not inherit from `java.lang.Object`.

- When viewing the Overview page, clicking on "Tree" displays the hierarchy for all packages.
- When viewing a particular package, class or interface page, clicking "Tree" displays the hierarchy for only that package.

Deprecated API

The [Deprecated API](#) page lists all of the API that have been deprecated. A deprecated API is not recommended for use, generally due to improvements, and a replacement API is usually given. Deprecated APIs may be removed in future implementations.

Index

The [Index](#) contains an alphabetic list of all classes, interfaces, constructors, methods, and fields.

Prev/Next

These links take you to the next or previous class, interface, package, or related page.

Frames/No Frames

These links show and hide the HTML frames. All pages are available with or without frames.

Serialized Form

Each serializable or externalizable class has a description of its serialization fields and methods. This information is of interest to re-implementors, not to developers using the API. While there is no link in the navigation bar, you can get to this information by going to any serialized class and clicking "Serialized Form" in the "See also" section of the class description.

This help file applies to API documentation generated using the standard doclet.

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) **Help**

[PREV](#) [NEXT](#)

[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

Overview Package Class Use [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV](#) [NEXT](#)

[FRAMES](#) [NO FRAMES](#)

Java API for XML Parsing 1.1.1

Packages

javax.xml.parsers	Provides classes allowing the processing of XML documents.
javax.xml.transform	This package defines the generic APIs for processing transformation instructions, and performing a transformation from source to result.
javax.xml.transform.dom	This package implements DOM-specific transformation APIs. The DOMSource class allows the client of the implementation of this API to specify a DOM Node as the source of the input tree.
javax.xml.transform.sax	This package implements SAX2-specific transformation APIs.
javax.xml.transform.stream	This package implements stream- and URI- specific transformation APIs.
org.w3c.dom	
org.xml.sax	
org.xml.sax.ext	
org.xml.sax.helpers	

Overview Package Class Use [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV](#) [NEXT](#)

[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Hierarchy For Package javax.xml.parsers

Package Hierarchies:

[All Packages](#)

Class Hierarchy

- class java.lang.Object
 - class javax.xml.parsers.[DocumentBuilder](#)
 - class javax.xml.parsers.[DocumentBuilderFactory](#)
 - class javax.xml.parsers.[SAXParser](#)
 - class javax.xml.parsers.[SAXParserFactory](#)
 - class java.lang.Throwable (implements java.io.Serializable)
 - class java.lang.Error
 - class javax.xml.parsers.[FactoryConfigurationError](#)
 - class java.lang.Exception
 - class javax.xml.parsers.[ParserConfigurationException](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Hierarchy For Package javax.xml.transform

Package Hierarchies:

[All Packages](#)

Class Hierarchy

- class java.lang.Object
 - class javax.xml.transform.[OutputKeys](#)
 - class java.lang.Throwable (implements java.io.Serializable)
 - class java.lang.Error
 - class javax.xml.transform.[TransformerFactoryConfigurationError](#)
 - class java.lang.Exception
 - class javax.xml.transform.[TransformerException](#)
 - class javax.xml.transform.[TransformerConfigurationException](#)
 - class javax.xml.transform.[Transformer](#)
 - class javax.xml.transform.[TransformerFactory](#)

Interface Hierarchy

- interface javax.xml.transform.[ErrorListener](#)
- interface javax.xml.transform.[Result](#)
- interface javax.xml.transform.[Source](#)
- interface javax.xml.transform.[SourceLocator](#)
- interface javax.xml.transform.[Templates](#)
- interface javax.xml.transform.[URIResolver](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)Submit Feedback to xml-feedback@java.sun.comCopyright 1998-2001 [Sun Microsystems Inc.](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Hierarchy For Package javax.xml.transform.dom

Package Hierarchies:

[All Packages](#)

Class Hierarchy

- class java.lang.Object
 - class javax.xml.transform.dom.[DOMResult](#) (implements javax.xml.transform.[Result](#))
 - class javax.xml.transform.dom.[DOMSource](#) (implements javax.xml.transform.[Source](#))

Interface Hierarchy

- interface javax.xml.transform.[SourceLocator](#)
 - interface javax.xml.transform.dom.[DOMLocator](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Hierarchy For Package javax.xml.transform.sax

Package Hierarchies:

[All Packages](#)

Class Hierarchy

- class java.lang.Object
 - class javax.xml.transform.sax.[SAXResult](#) (implements javax.xml.transform.[Result](#))
 - class javax.xml.transform.sax.[SAXSource](#) (implements javax.xml.transform.[Source](#))
 - class javax.xml.transform.[TransformerFactory](#)
 - class javax.xml.transform.sax.[SAXTransformerFactory](#)

Interface Hierarchy

- interface org.xml.sax.[ContentHandler](#)
 - interface javax.xml.transform.sax.[TemplatesHandler](#)
 - interface javax.xml.transform.sax.[TransformerHandler](#) (also extends org.xml.sax.[DTDHandler](#), org.xml.sax.ext.[LexicalHandler](#))
- interface org.xml.sax.[DTDHandler](#)
 - interface javax.xml.transform.sax.[TransformerHandler](#) (also extends org.xml.sax.[ContentHandler](#), org.xml.sax.ext.[LexicalHandler](#))
- interface org.xml.sax.ext.[LexicalHandler](#)
 - interface javax.xml.transform.sax.[TransformerHandler](#) (also extends org.xml.sax.[ContentHandler](#), org.xml.sax.[DTDHandler](#))

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)Submit Feedback to xml-feedback@java.sun.comCopyright 1998-2001 [Sun Microsystems Inc.](#)All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Hierarchy For Package javax.xml.transform.stream

Package Hierarchies:

[All Packages](#)

Class Hierarchy

- class java.lang.Object
 - class javax.xml.transform.stream.[StreamResult](#) (implements javax.xml.transform.[Result](#))
 - class javax.xml.transform.stream.[StreamSource](#) (implements javax.xml.transform.[Source](#))

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Hierarchy For Package org.w3c.dom

Package Hierarchies:

[All Packages](#)

Class Hierarchy

- class java.lang.Object
 - class java.lang.Throwable (implements java.io.Serializable)
 - class java.lang.Exception
 - class java.lang.RuntimeException
 - class org.w3c.dom.[DOMException](#)

Interface Hierarchy

- interface org.w3c.dom.[DOMImplementation](#)
- interface org.w3c.dom.[NamedNodeMap](#)
- interface org.w3c.dom.[Node](#)
 - interface org.w3c.dom.[Attr](#)
 - interface org.w3c.dom.[CharacterData](#)
 - interface org.w3c.dom.[Comment](#)
 - interface org.w3c.dom.[Text](#)
 - interface org.w3c.dom.[CDATASection](#)
 - interface org.w3c.dom.[Document](#)
 - interface org.w3c.dom.[DocumentFragment](#)
 - interface org.w3c.dom.[DocumentType](#)
 - interface org.w3c.dom.[Element](#)
 - interface org.w3c.dom.[Entity](#)
 - interface org.w3c.dom.[EntityReference](#)
 - interface org.w3c.dom.[Notation](#)
 - interface org.w3c.dom.[ProcessingInstruction](#)

- interface org.w3c.dom.[NodeList](#)

[Overview](#) [Package](#) Class Use **Tree** [Deprecated](#) [Index](#) [Help](#)

[PREV](#) [NEXT](#)

[FRAMES](#) [NO FRAMES](#)

Hierarchy For Package org.xml.sax

Package Hierarchies:

[All Packages](#)

Class Hierarchy

- class java.lang.Object
 - class org.xml.sax.[HandlerBase](#) (implements org.xml.sax.[DocumentHandler](#), org.xml.sax.[DTDHandler](#), org.xml.sax.[EntityResolver](#), org.xml.sax.[ErrorHandler](#))
 - class org.xml.sax.[InputSource](#)
 - class java.lang.Throwable (implements java.io.Serializable)
 - class java.lang.Exception
 - class org.xml.sax.[SAXException](#)
 - class org.xml.sax.[SAXNotRecognizedException](#)
 - class org.xml.sax.[SAXNotSupportedException](#)
 - class org.xml.sax.[SAXParseException](#)

Interface Hierarchy

- interface org.xml.sax.[AttributeList](#)
- interface org.xml.sax.[Attributes](#)
- interface org.xml.sax.[ContentHandler](#)
- interface org.xml.sax.[DocumentHandler](#)
- interface org.xml.sax.[DTDHandler](#)
- interface org.xml.sax.[EntityResolver](#)
- interface org.xml.sax.[ErrorHandler](#)
- interface org.xml.sax.[Locator](#)
- interface org.xml.sax.[Parser](#)
- interface org.xml.sax.[XMLReader](#)
 - interface org.xml.sax.[XMLFilter](#)

[Overview](#) [Package](#) [Class](#) [Use](#) **Tree** [Deprecated](#) [Index](#) [Help](#)

[PREV](#) [NEXT](#)

[FRAMES](#) [NO FRAMES](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Hierarchy For Package org.xml.sax.ext

Package Hierarchies:

[All Packages](#)

Interface Hierarchy

- interface org.xml.sax.ext.[DeclHandler](#)
 - interface org.xml.sax.ext.[LexicalHandler](#)
-

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Hierarchy For Package org.xml.sax.helpers

Package Hierarchies:

[All Packages](#)

Class Hierarchy

- class java.lang.Object
 - class org.xml.sax.helpers.[AttributeListImpl](#) (implements org.xml.sax.[AttributeList](#))
 - class org.xml.sax.helpers.[AttributesImpl](#) (implements org.xml.sax.[Attributes](#))
 - class org.xml.sax.helpers.[DefaultHandler](#) (implements org.xml.sax.[ContentHandler](#), org.xml.sax.[DTDHandler](#), org.xml.sax.[EntityResolver](#), org.xml.sax.[ErrorHandler](#))
 - class org.xml.sax.helpers.[LocatorImpl](#) (implements org.xml.sax.[Locator](#))
 - class org.xml.sax.helpers.[NamespaceSupport](#)
 - class org.xml.sax.helpers.[ParserAdapter](#) (implements org.xml.sax.[DocumentHandler](#), org.xml.sax.[XMLReader](#))
 - class org.xml.sax.helpers.[ParserFactory](#)
 - class org.xml.sax.helpers.[XMLFilterImpl](#) (implements org.xml.sax.[ContentHandler](#), org.xml.sax.[DTDHandler](#), org.xml.sax.[EntityResolver](#), org.xml.sax.[ErrorHandler](#), org.xml.sax.[XMLFilter](#))
 - class org.xml.sax.helpers.[XMLReaderAdapter](#) (implements org.xml.sax.[ContentHandler](#), org.xml.sax.[Parser](#))
 - class org.xml.sax.helpers.[XMLReaderFactory](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax.helpers

Class AttributeListImpl

java.lang.Object

```

|
+--org.xml.sax.helpers.AttributeListImpl

```

Deprecated. *This class implements a deprecated interface, [AttributeList](#); that interface has been replaced by [Attributes](#), which is implemented in the [AttributesImpl](#) helper class.*

public class **AttributeListImpl**

extends java.lang.Object

implements [AttributeList](#)

Default implementation for AttributeList.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

AttributeList implements the deprecated SAX1 [AttributeList](#) interface, and has been replaced by the new SAX2 [AttributesImpl](#) interface.

This class provides a convenience implementation of the SAX [AttributeList](#) interface. This implementation is useful both for SAX parser writers, who can use it to provide attributes to the application, and for SAX application writers, who can use it to create a persistent copy of an element's attribute specifications:

```

private AttributeList myatts;

public void startElement (String name, AttributeList atts)
{
    // create a persistent copy of the attribute list
    // for use outside this method
    myatts = new AttributeListImpl(atts);
    [...]
}

```

Please note that SAX parsers are not required to use this class to provide an implementation of AttributeList; it is supplied only as an optional convenience. In particular, parser writers are encouraged to invent more efficient implementations.

Since:

SAX 1.0

Version:

2.0

Author:David Megginson, sax@megginson.com**See Also:**

[AttributeList](#), [DocumentHandler.startElement\(java.lang.String, org.xml.sax.AttributeList\)](#)

Constructor Summary

[AttributeListImpl](#)()

Deprecated. Create an empty attribute list.

[AttributeListImpl](#)([AttributeList](#) atts)

Deprecated. Construct a persistent copy of an existing attribute list.

Method Summary

void	addAttribute (java.lang.String name, java.lang.String type, java.lang.String value) Deprecated. Add an attribute to an attribute list.
------	---

void	clear () Deprecated. Clear the attribute list.
------	---

int	getLength () Deprecated. Return the number of attributes in the list.
-----	--

java.lang.String	getName (int i) Deprecated. Get the name of an attribute (by position).
------------------	---

java.lang.String	getType (int i) Deprecated. Get the type of an attribute (by position).
------------------	---

java.lang.String	getType (java.lang.String name) Deprecated. Get the type of an attribute (by name).
------------------	---

java.lang.String	getValue (int i) Deprecated. Get the value of an attribute (by position).
------------------	---

java.lang.String	getValue (java.lang.String name) Deprecated. Get the value of an attribute (by name).
------------------	---

void	removeAttribute (java.lang.String name) Deprecated. Remove an attribute from the list.
void	setAttributeList (AttributeList atts) Deprecated. Set the attribute list, discarding previous contents.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AttributeListImpl

```
public AttributeListImpl()
```

Deprecated.

Create an empty attribute list.

This constructor is most useful for parser writers, who will use it to create a single, reusable attribute list that can be reset with the clear method between elements.

See Also:

[addAttribute\(java.lang.String, java.lang.String, java.lang.String\)](#), [clear\(\)](#)

AttributeListImpl

```
public AttributeListImpl(AttributeList atts)
```

Deprecated.

Construct a persistent copy of an existing attribute list.

This constructor is most useful for application writers, who will use it to create a persistent copy of an existing attribute list.

Parameters:

atts - The attribute list to copy

See Also:

[DocumentHandler.startElement\(java.lang.String, org.xml.sax.AttributeList\)](#)

Method Detail

setAttributeList

```
public void setAttributeList(AttributeList atts)
```

Deprecated.

Set the attribute list, discarding previous contents.

This method allows an application writer to reuse an attribute list easily.

Parameters:

atts - The attribute list to copy.

addAttribute

```
public void addAttribute(java.lang.String name,  
                           java.lang.String type,  
                           java.lang.String value)
```

Deprecated.

Add an attribute to an attribute list.

This method is provided for SAX parser writers, to allow them to build up an attribute list incrementally before delivering it to the application.

Parameters:

name - The attribute name.

type - The attribute type ("NM_TOKEN" for an enumeration).

value - The attribute value (must not be null).

See Also:

[removeAttribute\(java.lang.String\)](#),
[DocumentHandler.startElement\(java.lang.String,
org.xml.sax.AttributeList\)](#)

removeAttribute

```
public void removeAttribute(java.lang.String name)
```

Deprecated.

Remove an attribute from the list.

SAX application writers can use this method to filter an attribute out of an `AttributeList`. Note that invoking this method will change the length of the attribute list and some of the attribute's indices.

If the requested attribute is not in the list, this is a no-op.

Parameters:

name - The attribute name.

See Also:

[addAttribute\(java.lang.String, java.lang.String, java.lang.String\)](#)

clear

```
public void clear()
```

Deprecated.

Clear the attribute list.

SAX parser writers can use this method to reset the attribute list between `DocumentHandler.startElement` events. Normally, it will make sense to reuse the same `AttributeListImpl` object rather than allocating a new one each time.

See Also:

[DocumentHandler.startElement\(java.lang.String, org.xml.sax.AttributeList\)](#)

getLength

```
public int getLength()
```

Deprecated.

Return the number of attributes in the list.

Specified by:

[getLength](#) in interface [AttributeList](#)

Returns:

The number of attributes in the list.

See Also:

[AttributeList.getLength\(\)](#)

`getName`

```
public java.lang.String getName(int i)
```

Deprecated.

Get the name of an attribute (by position).

Specified by:

[`getName`](#) in interface [`AttributeList`](#)

Parameters:

`i` - The position of the attribute in the list.

Returns:

The attribute name as a string, or null if there is no attribute at that position.

See Also:

[`AttributeList.getName\(int\)`](#)

`getType`

```
public java.lang.String getType(int i)
```

Deprecated.

Get the type of an attribute (by position).

Specified by:

[`getType`](#) in interface [`AttributeList`](#)

Parameters:

`i` - The position of the attribute in the list.

Returns:

The attribute type as a string ("NMTOKEN" for an enumeration, and "CDATA" if no declaration was read), or null if there is no attribute at that position.

See Also:

[`AttributeList.getType\(int\)`](#)

`getValue`

```
public java.lang.String getValue(int i)
```

Deprecated.

Get the value of an attribute (by position).

Specified by:

[getValue](#) in interface [AttributeList](#)

Parameters:

i - The position of the attribute in the list.

Returns:

The attribute value as a string, or null if there is no attribute at that position.

See Also:

[AttributeList.getValue\(int\)](#)

getType

```
public java.lang.String getType(java.lang.String name)
```

Deprecated.

Get the type of an attribute (by name).

Specified by:

[getType](#) in interface [AttributeList](#)

Parameters:

name - The attribute name.

Returns:

The attribute type as a string ("NMTOKEN" for an enumeration, and "CDATA" if no declaration was read).

See Also:

[AttributeList.getType\(java.lang.String\)](#)

getValue

```
public java.lang.String getValue(java.lang.String name)
```

Deprecated.

Get the value of an attribute (by name).

Specified by:

[getValue](#) in interface [AttributeList](#)

Parameters:

name - The attribute name.

See Also:

[AttributeList.getValue\(java.lang.String\)](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: INNER | FIELD | [CONSTR](#) | [METHOD](#)

DETAIL: FIELD | [CONSTR](#) | [METHOD](#)

`org.xml.sax`

Interface `AttributeList`

All Known Implementing Classes:

[AttributeListImpl](#)

Deprecated. *This interface has been replaced by the SAX2 [Attributes](#) interface, which includes Namespace support.*

public abstract interface **AttributeList**

Interface for an element's attribute specifications.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This is the original SAX1 interface for reporting an element's attributes. Unlike the new [Attributes](#) interface, it does not support Namespace-related information.

When an attribute list is supplied as part of a [startElement](#) event, the list will return valid results only during the scope of the event; once the event handler returns control to the parser, the attribute list is invalid. To save a persistent copy of the attribute list, use the SAX1 [AttributeListImpl](#) helper class.

An attribute list includes only attributes that have been specified or defaulted: #IMPLIED attributes will not be included.

There are two ways for the SAX application to obtain information from the `AttributeList`. First, it can iterate through the entire list:

```
public void startElement (String name, AttributeList atts) {
    for (int i = 0; i < atts.getLength(); i++) {
        String name = atts.getName(i);
        String type = atts.getType(i);
        String value = atts.getValue(i);
        [...]
    }
}
```

(Note that the result of `getLength()` will be zero if there are no attributes.)

As an alternative, the application can request the value or type of specific attributes:

```
public void startElement (String name, AttributeList atts) {
```

```

String identifier = atts.getValue("id");
String label = atts.getValue("label");
[...]
}

```

Since:

SAX 1.0

Version:

2.0

Author:David Megginson, sax@megginson.com**See Also:**[startElement](#), [AttributeListImpl](#)

Method Summary

int	getLength () Deprecated. Return the number of attributes in this list.
java.lang.String	getName (int i) Deprecated. Return the name of an attribute in this list (by position).
java.lang.String	getType (int i) Deprecated. Return the type of an attribute in the list (by position).
java.lang.String	getType (java.lang.String name) Deprecated. Return the type of an attribute in the list (by name).
java.lang.String	getValue (int i) Deprecated. Return the value of an attribute in the list (by position).
java.lang.String	getValue (java.lang.String name) Deprecated. Return the value of an attribute in the list (by name).

Method Detail

getLength

```
public int getLength( )
```

Deprecated.

Return the number of attributes in this list.

The SAX parser may provide attributes in any arbitrary order, regardless of the order in which they

were declared or specified. The number of attributes may be zero.

Returns:

The number of attributes in the list.

getName

```
public java.lang.String getName(int i)
```

Deprecated.

Return the name of an attribute in this list (by position).

The names must be unique: the SAX parser shall not include the same attribute twice. Attributes without values (those declared #IMPLIED without a value specified in the start tag) will be omitted from the list.

If the attribute name has a namespace prefix, the prefix will still be attached.

Parameters:

i - The index of the attribute in the list (starting at 0).

Returns:

The name of the indexed attribute, or null if the index is out of range.

See Also:

[getLength\(\)](#)

getType

```
public java.lang.String getType(int i)
```

Deprecated.

Return the type of an attribute in the list (by position).

The attribute type is one of the strings "CDATA", "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", "ENTITIES", or "NOTATION" (always in upper case).

If the parser has not read a declaration for the attribute, or if the parser does not report attribute types, then it must return the value "CDATA" as stated in the XML 1.0 Recommendation (clause 3.3.3, "Attribute-Value Normalization").

For an enumerated attribute that is not a notation, the parser will report the type as "NMTOKEN".

Parameters:

i - The index of the attribute in the list (starting at 0).

Returns:

The attribute type as a string, or null if the index is out of range.

See Also:

[getLength\(\)](#), [getType\(java.lang.String\)](#)

getValue

```
public java.lang.String getValue(int i)
```

Deprecated.

Return the value of an attribute in the list (by position).

If the attribute value is a list of tokens (IDREFS, ENTITIES, or NMTOKENS), the tokens will be concatenated into a single string separated by whitespace.

Parameters:

`i` - The index of the attribute in the list (starting at 0).

Returns:

The attribute value as a string, or null if the index is out of range.

See Also:

[getLength\(\)](#), [getValue\(java.lang.String\)](#)

getType

```
public java.lang.String getType(java.lang.String name)
```

Deprecated.

Return the type of an attribute in the list (by name).

The return value is the same as the return value for `getType(int)`.

If the attribute name has a namespace prefix in the document, the application must include the prefix here.

Parameters:

`name` - The name of the attribute.

Returns:

The attribute type as a string, or null if no such attribute exists.

See Also:

[getType\(int\)](#)

getValue

```
public java.lang.String getValue(java.lang.String name)
```

Deprecated.

Return the value of an attribute in the list (by name).

The return value is the same as the return value for `getValue(int)`.

If the attribute name has a namespace prefix in the document, the application must include the prefix here.

Parameters:

`i` - The index of the attribute in the list.

Returns:

The attribute value as a string, or null if no such attribute exists.

See Also:

[getValue\(int\)](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax.helpers

Class AttributesImpl

java.lang.Object

```

|
+--org.xml.sax.helpers.AttributesImpl

```

public class **AttributesImpl**

extends java.lang.Object

implements [Attributes](#)

Default implementation of the Attributes interface.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This class provides a default implementation of the SAX2 [Attributes](#) interface, with the addition of manipulators so that the list can be modified or reused.

There are two typical uses of this class:

1. to take a persistent snapshot of an Attributes object in a [startElement](#) event; or
2. to construct or modify an Attributes object in a SAX2 driver or filter.

This class replaces the now-deprecated SAX1 [AttributeListImpl](#) class; in addition to supporting the updated Attributes interface rather than the deprecated [AttributeList](#) interface, it also includes a much more efficient implementation using a single array rather than a set of Vectors.

Since:

SAX 2.0

Version:

2.0

Author:David Megginson, sax@megginson.com

Constructor Summary

[AttributesImpl](#)()

Construct a new, empty AttributesImpl object.

[AttributesImpl](#)([Attributes](#) atts)

Copy an existing Attributes object.

Method Summary

void	addAttribute (java.lang.String uri, java.lang.String localName, java.lang.String qName, java.lang.String type, java.lang.String value) Add an attribute to the end of the list.
void	clear () Clear the attribute list for reuse.
int	getIndex (java.lang.String qName) Look up an attribute's index by qualified (prefixed) name.
int	getIndex (java.lang.String uri, java.lang.String localName) Look up an attribute's index by Namespace name.
int	getLength () Return the number of attributes in the list.
java.lang.String	getLocalName (int index) Return an attribute's local name.
java.lang.String	getQName (int index) Return an attribute's qualified (prefixed) name.
java.lang.String	getType (int index) Return an attribute's type by index.
java.lang.String	getType (java.lang.String qName) Look up an attribute's type by qualified (prefixed) name.
java.lang.String	getType (java.lang.String uri, java.lang.String localName) Look up an attribute's type by Namespace-qualified name.
java.lang.String	getURI (int index) Return an attribute's Namespace URI.
java.lang.String	getValue (int index) Return an attribute's value by index.
java.lang.String	getValue (java.lang.String qName) Look up an attribute's value by qualified (prefixed) name.

java.lang.String	<code>getValue</code> (java.lang.String uri, java.lang.String localName) Look up an attribute's value by Namespace-qualified name.
void	<code>removeAttribute</code> (int index) Remove an attribute from the list.
void	<code>setAttribute</code> (int index, java.lang.String uri, java.lang.String localName, java.lang.String qName, java.lang.String type, java.lang.String value) Set an attribute in the list.
void	<code>setAttributes</code> (<code>Attributes</code> atts) Copy an entire Attributes object.
void	<code>setLocalName</code> (int index, java.lang.String localName) Set the local name of a specific attribute.
void	<code>setQName</code> (int index, java.lang.String qName) Set the qualified name of a specific attribute.
void	<code>setType</code> (int index, java.lang.String type) Set the type of a specific attribute.
void	<code>setURI</code> (int index, java.lang.String uri) Set the Namespace URI of a specific attribute.
void	<code>setValue</code> (int index, java.lang.String value) Set the value of a specific attribute.

Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`,
`toString`, `wait`, `wait`, `wait`

Constructor Detail

AttributesImpl

```
public AttributesImpl()
```

Construct a new, empty AttributesImpl object.

AttributesImpl

```
public AttributesImpl(Attributes atts)
```

Copy an existing Attributes object.

This constructor is especially useful inside a [startElement](#) event.

Parameters:

atts - The existing Attributes object.

Method Detail

getLength

```
public int getLength()
```

Return the number of attributes in the list.

Specified by:

[getLength](#) in interface [Attributes](#)

Returns:

The number of attributes in the list.

See Also:

[Attributes.getLength\(\)](#)

getURI

```
public java.lang.String getURI(int index)
```

Return an attribute's Namespace URI.

Specified by:

[getURI](#) in interface [Attributes](#)

Parameters:

index - The attribute's index (zero-based).

Returns:

The Namespace URI, the empty string if none is available, or null if the index is out of range.

See Also:

[Attributes.getURI\(int\)](#)

getLocalName

```
public java.lang.String getLocalName(int index)
```

Return an attribute's local name.

Specified by:

[getLocalName](#) in interface [Attributes](#)

Parameters:

`index` - The attribute's index (zero-based).

Returns:

The attribute's local name, the empty string if none is available, or null if the index is out of range.

See Also:

[Attributes.getLocalName\(int\)](#)

getName

```
public java.lang.String getName(int index)
```

Return an attribute's qualified (prefixed) name.

Specified by:

[getName](#) in interface [Attributes](#)

Parameters:

`index` - The attribute's index (zero-based).

Returns:

The attribute's qualified name, the empty string if none is available, or null if the index is out of bounds.

See Also:

[Attributes.getName\(int\)](#)

getType

```
public java.lang.String getType(int index)
```

Return an attribute's type by index.

Specified by:

[getType](#) in interface [Attributes](#)

Parameters:

`index` - The attribute's index (zero-based).

Returns:

The attribute's type, "CDATA" if the type is unknown, or null if the index is out of bounds.

See Also:

[Attributes.getType\(int\)](#)

getValue

```
public java.lang.String getValue(int index)
```

Return an attribute's value by index.

Specified by:

[getValue](#) in interface [Attributes](#)

Parameters:

`index` - The attribute's index (zero-based).

Returns:

The attribute's value or null if the index is out of bounds.

See Also:

[Attributes.getValue\(int\)](#)

getIndex

```
public int getIndex(java.lang.String uri,  
                    java.lang.String localName)
```

Look up an attribute's index by Namespace name.

In many cases, it will be more efficient to look up the name once and use the index query methods rather than using the name query methods repeatedly.

Specified by:

[getIndex](#) in interface [Attributes](#)

Parameters:

`uri` - The attribute's Namespace URI, or the empty string if none is available.

`localName` - The attribute's local name.

Returns:

The attribute's index, or -1 if none matches.

See Also:

[Attributes.getIndex\(java.lang.String,java.lang.String\)](#)

getIndex

```
public int getIndex(java.lang.String qName)
```

Look up an attribute's index by qualified (prefixed) name.

Specified by:

[getIndex](#) in interface [Attributes](#)

Parameters:

qName - The qualified name.

Returns:

The attribute's index, or -1 if none matches.

See Also:

[Attributes.getIndex\(java.lang.String\)](#)

getType

```
public java.lang.String getType(java.lang.String uri,  
                                java.lang.String localName)
```

Look up an attribute's type by Namespace-qualified name.

Specified by:

[getType](#) in interface [Attributes](#)

Parameters:

uri - The Namespace URI, or the empty string for a name with no explicit Namespace URI.

localName - The local name.

Returns:

The attribute's type, or null if there is no matching attribute.

See Also:

[Attributes.getType\(java.lang.String,java.lang.String\)](#)

getType

```
public java.lang.String getType(java.lang.String qName)
```

Look up an attribute's type by qualified (prefixed) name.

Specified by:

[getType](#) in interface [Attributes](#)

Parameters:

qName - The qualified name.

Returns:

The attribute's type, or null if there is no matching attribute.

See Also:

[Attributes.getType\(java.lang.String\)](#)

getValue

```
public java.lang.String getValue(java.lang.String uri,  
                                   java.lang.String localName)
```

Look up an attribute's value by Namespace-qualified name.

Specified by:

[getValue](#) in interface [Attributes](#)

Parameters:

uri - The Namespace URI, or the empty string for a name with no explicit Namespace URI.

localName - The local name.

Returns:

The attribute's value, or null if there is no matching attribute.

See Also:

[Attributes.getValue\(java.lang.String,java.lang.String\)](#)

getValue

```
public java.lang.String getValue(java.lang.String qName)
```

Look up an attribute's value by qualified (prefixed) name.

Specified by:

[getValue](#) in interface [Attributes](#)

Parameters:

qName - The qualified name.

Returns:

The attribute's value, or null if there is no matching attribute.

See Also:

[Attributes.getValue\(java.lang.String\)](#)

clear

```
public void clear()
```

Clear the attribute list for reuse.

Note that no memory is actually freed by this call: the current arrays are kept so that they can be reused.

setAttributes

```
public void setAttributes(Attributes atts)
```

Copy an entire Attributes object.

It may be more efficient to reuse an existing object rather than constantly allocating new ones.

Parameters:

`atts` - The attributes to copy.

addAttribute

```
public void addAttribute(java.lang.String uri,  
                        java.lang.String localName,  
                        java.lang.String qName,  
                        java.lang.String type,  
                        java.lang.String value)
```

Add an attribute to the end of the list.

For the sake of speed, this method does no checking to see if the attribute is already in the list: that is the responsibility of the application.

Parameters:

`uri` - The Namespace URI, or the empty string if none is available or Namespace processing is not being performed.

`localName` - The local name, or the empty string if Namespace processing is not being performed.

`qName` - The qualified (prefixed) name, or the empty string if qualified names are not

available.

`type` - The attribute type as a string.

`value` - The attribute value.

setAttribute

```
public void setAttribute(int index,
                          java.lang.String uri,
                          java.lang.String localName,
                          java.lang.String qName,
                          java.lang.String type,
                          java.lang.String value)
```

Set an attribute in the list.

For the sake of speed, this method does no checking for name conflicts or well-formedness: such checks are the responsibility of the application.

Parameters:

`index` - The index of the attribute (zero-based).

`uri` - The Namespace URI, or the empty string if none is available or Namespace processing is not being performed.

`localName` - The local name, or the empty string if Namespace processing is not being performed.

`qName` - The qualified name, or the empty string if qualified names are not available.

`type` - The attribute type as a string.

`value` - The attribute value.

Throws:

`java.lang.ArrayIndexOutOfBoundsException` - When the supplied index does not point to an attribute in the list.

removeAttribute

```
public void removeAttribute(int index)
```

Remove an attribute from the list.

Parameters:

`index` - The index of the attribute (zero-based).

Throws:

`java.lang.ArrayIndexOutOfBoundsException` - When the supplied index does not point to an attribute in the list.

setURI

```
public void setURI(int index,  
                   java.lang.String uri)
```

Set the Namespace URI of a specific attribute.

Parameters:

`index` - The index of the attribute (zero-based).

`uri` - The attribute's Namespace URI, or the empty string for none.

Throws:

`java.lang.ArrayIndexOutOfBoundsException` - When the supplied index does not point to an attribute in the list.

setLocalName

```
public void setLocalName(int index,  
                          java.lang.String localName)
```

Set the local name of a specific attribute.

Parameters:

`index` - The index of the attribute (zero-based).

`localName` - The attribute's local name, or the empty string for none.

Throws:

`java.lang.ArrayIndexOutOfBoundsException` - When the supplied index does not point to an attribute in the list.

setQName

```
public void setQName(int index,  
                     java.lang.String qName)
```

Set the qualified name of a specific attribute.

Parameters:

`index` - The index of the attribute (zero-based).

`qName` - The attribute's qualified name, or the empty string for none.

Throws:

`java.lang.ArrayIndexOutOfBoundsException` - When the supplied index does not point to an attribute in the list.

setType

```
public void setType(int index,  
                    java.lang.String type)
```

Set the type of a specific attribute.

Parameters:

`index` - The index of the attribute (zero-based).

`type` - The attribute's type.

Throws:

`java.lang.ArrayIndexOutOfBoundsException` - When the supplied index does not point to an attribute in the list.

setValue

```
public void setValue(int index,  
                     java.lang.String value)
```

Set the value of a specific attribute.

Parameters:

`index` - The index of the attribute (zero-based).

`value` - The attribute's value.

Throws:

`java.lang.ArrayIndexOutOfBoundsException` - When the supplied index does not point to an attribute in the list.

[Overview](#) [Package](#) **[Class](#)** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Interface Attributes

All Known Implementing Classes:

[AttributesImpl](#)

public abstract interface **Attributes**

Interface for a list of XML attributes.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This interface allows access to a list of attributes in three different ways:

1. by attribute index;
2. by Namespace-qualified name; or
3. by qualified (prefixed) name.

The list will not contain attributes that were declared #IMPLIED but not specified in the start tag. It will also not contain attributes used as Namespace declarations (xmlns*) unless the `http://xml.org/sax/features/namespace-prefixes` feature is set to true (it is false by default).

If the namespace-prefixes feature (see above) is false, access by qualified name may not be available; if the `http://xml.org/sax/features/namespaces` feature is false, access by Namespace-qualified names may not be available.

This interface replaces the now-deprecated SAX1 [AttributeList](#) interface, which does not contain Namespace support. In addition to Namespace support, it adds the `getIndex` methods (below).

The order of attributes in the list is unspecified, and will vary from implementation to implementation.

Since:

SAX 2.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[AttributeListImpl](#)

Method Summary

int	<code>getIndex</code> (java.lang.String qName) Look up the index of an attribute by XML 1.0 qualified name.
int	<code>getIndex</code> (java.lang.String uri, java.lang.String localPart) Look up the index of an attribute by Namespace name.
int	<code>getLength</code> () Return the number of attributes in the list.
java.lang.String	<code>getLocalName</code> (int index) Look up an attribute's local name by index.
java.lang.String	<code>getQName</code> (int index) Look up an attribute's XML 1.0 qualified name by index.
java.lang.String	<code>getType</code> (int index) Look up an attribute's type by index.
java.lang.String	<code>getType</code> (java.lang.String qName) Look up an attribute's type by XML 1.0 qualified name.
java.lang.String	<code>getType</code> (java.lang.String uri, java.lang.String localName) Look up an attribute's type by Namespace name.
java.lang.String	<code>getURI</code> (int index) Look up an attribute's Namespace URI by index.
java.lang.String	<code>getValue</code> (int index) Look up an attribute's value by index.
java.lang.String	<code>getValue</code> (java.lang.String qName) Look up an attribute's value by XML 1.0 qualified name.
java.lang.String	<code>getValue</code> (java.lang.String uri, java.lang.String localName) Look up an attribute's value by Namespace name.

Method Detail

getLength

```
public int getLength()
```

Return the number of attributes in the list.

Once you know the number of attributes, you can iterate through the list.

Returns:

The number of attributes in the list.

See Also:

[getURI\(int\)](#), [getLocalName\(int\)](#), [getQName\(int\)](#), [getType\(int\)](#),
[getValue\(int\)](#)

getURI

```
public java.lang.String getURI(int index)
```

Look up an attribute's Namespace URI by index.

Parameters:

`index` - The attribute index (zero-based).

Returns:

The Namespace URI, or the empty string if none is available, or null if the index is out of range.

See Also:

[getLength\(\)](#)

getLocalName

```
public java.lang.String getLocalName(int index)
```

Look up an attribute's local name by index.

Parameters:

`index` - The attribute index (zero-based).

Returns:

The local name, or the empty string if Namespace processing is not being performed, or null if the index is out of range.

See Also:

[getLength\(\)](#)

getQName

```
public java.lang.String getQName(int index)
```

Look up an attribute's XML 1.0 qualified name by index.

Parameters:

`index` - The attribute index (zero-based).

Returns:

The XML 1.0 qualified name, or the empty string if none is available, or null if the index is out of range.

See Also:

[`getLength\(\)`](#)

getType

```
public java.lang.String getType(int index)
```

Look up an attribute's type by index.

The attribute type is one of the strings "CDATA", "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", "ENTITIES", or "NOTATION" (always in upper case).

If the parser has not read a declaration for the attribute, or if the parser does not report attribute types, then it must return the value "CDATA" as stated in the XML 1.0 Recommendation (clause 3.3.3, "Attribute-Value Normalization").

For an enumerated attribute that is not a notation, the parser will report the type as "NMTOKEN".

Parameters:

`index` - The attribute index (zero-based).

Returns:

The attribute's type as a string, or null if the index is out of range.

See Also:

[`getLength\(\)`](#)

getValue

```
public java.lang.String getValue(int index)
```

Look up an attribute's value by index.

If the attribute value is a list of tokens (IDREFS, ENTITIES, or NMTOKENS), the tokens will be concatenated into a single string with each token separated by a single space.

Parameters:

`index` - The attribute index (zero-based).

Returns:

The attribute's value as a string, or null if the index is out of range.

See Also:

[getLength\(\)](#)

getIndex

```
public int getIndex(java.lang.String uri,  
                    java.lang.String localPart)
```

Look up the index of an attribute by Namespace name.

Parameters:

uri - The Namespace URI, or the empty string if the name has no Namespace URI.

localName - The attribute's local name.

Returns:

The index of the attribute, or -1 if it does not appear in the list.

getIndex

```
public int getIndex(java.lang.String qName)
```

Look up the index of an attribute by XML 1.0 qualified name.

Parameters:

qName - The qualified (prefixed) name.

Returns:

The index of the attribute, or -1 if it does not appear in the list.

getType

```
public java.lang.String getType(java.lang.String uri,  
                                java.lang.String localName)
```

Look up an attribute's type by Namespace name.

See [getType\(int\)](#) for a description of the possible types.

Parameters:

uri - The Namespace URI, or the empty String if the name has no Namespace URI.

localName - The local name of the attribute.

Returns:

The attribute type as a string, or null if the attribute is not in the list or if Namespace processing is not being performed.

getType

```
public java.lang.String getType(java.lang.String qName)
```

Look up an attribute's type by XML 1.0 qualified name.

See [getType\(int\)](#) for a description of the possible types.

Parameters:

qName - The XML 1.0 qualified name.

Returns:

The attribute type as a string, or null if the attribute is not in the list or if qualified names are not available.

getValue

```
public java.lang.String getValue(java.lang.String uri,  
                                   java.lang.String localName)
```

Look up an attribute's value by Namespace name.

See [getValue\(int\)](#) for a description of the possible values.

Parameters:

uri - The Namespace URI, or the empty String if the name has no Namespace URI.

localName - The local name of the attribute.

Returns:

The attribute value as a string, or null if the attribute is not in the list.

getValue

```
public java.lang.String getValue(java.lang.String qName)
```

Look up an attribute's value by XML 1.0 qualified name.

See [getValue\(int\)](#) for a description of the possible values.

Parameters:

qName - The XML 1.0 qualified name.

Returns:

The attribute value as a string, or null if the attribute is not in the list or if qualified names are not available.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: INNER | FIELD | CONSTR | [METHOD](#)

DETAIL: FIELD | CONSTR | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax.helpers

Class DefaultHandler

java.lang.Object

```

|
+--org.xml.sax.helpers.DefaultHandler

```

public class **DefaultHandler**

extends java.lang.Object

implements [EntityResolver](#), [DTDHandler](#), [ContentHandler](#), [ErrorHandler](#)

Default base class for SAX2 event handlers.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This class is available as a convenience base class for SAX2 applications: it provides default implementations for all of the callbacks in the four core SAX2 handler classes:

- [EntityResolver](#)
- [DTDHandler](#)
- [ContentHandler](#)
- [ErrorHandler](#)

Application writers can extend this class when they need to implement only part of an interface; parser writers can instantiate this class to provide default handlers when the application has not supplied its own.

This class replaces the deprecated SAX1 [HandlerBase](#) class.

Since:

SAX 2.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[EntityResolver](#), [DTDHandler](#), [ContentHandler](#), [ErrorHandler](#)

Constructor Summary

[DefaultHandler](#)()

Method Summary

void	characters (char[] ch, int start, int length) Receive notification of character data inside an element.
void	endDocument () Receive notification of the end of the document.
void	endElement (java.lang.String uri, java.lang.String localName, java.lang.String qName) Receive notification of the end of an element.
void	endPrefixMapping (java.lang.String prefix) Receive notification of the end of a Namespace mapping.
void	error (SAXParseException e) Receive notification of a recoverable parser error.
void	fatalError (SAXParseException e) Report a fatal XML parsing error.
void	ignorableWhitespace (char[] ch, int start, int length) Receive notification of ignorable whitespace in element content.
void	notationDecl (java.lang.String name, java.lang.String publicId, java.lang.String systemId) Receive notification of a notation declaration.
void	processingInstruction (java.lang.String target, java.lang.String data) Receive notification of a processing instruction.
InputSource	resolveEntity (java.lang.String publicId, java.lang.String systemId) Resolve an external entity.
void	setDocumentLocator (Locator locator) Receive a Locator object for document events.
void	skippedEntity (java.lang.String name) Receive notification of a skipped entity.
void	startDocument () Receive notification of the beginning of the document.

void	startElement (java.lang.String uri, java.lang.String localName, java.lang.String qName, Attributes attributes) Receive notification of the start of an element.
void	startPrefixMapping (java.lang.String prefix, java.lang.String uri) Receive notification of the start of a Namespace mapping.
void	unparsedEntityDecl (java.lang.String name, java.lang.String publicId, java.lang.String systemId, java.lang.String notationName) Receive notification of an unparsed entity declaration.
void	warning (SAXParseException e) Receive notification of a parser warning.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DefaultHandler

```
public DefaultHandler()
```

Method Detail

resolveEntity

```
public InputSource resolveEntity(java.lang.String publicId,  
                                java.lang.String systemId)  
    throws SAXException
```

Resolve an external entity.

Always return null, so that the parser will use the system identifier provided in the XML document. This method implements the SAX default behaviour: application writers can override it in a subclass to do special translations such as catalog lookups or URI redirection.

Specified by:

[resolveEntity](#) in interface [EntityResolver](#)

Parameters:

`publicId` - The public identifier, or null if none is available.

`systemId` - The system identifier provided in the XML document.

Returns:

The new input source, or null to require the default behaviour.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[EntityResolver.resolveEntity\(java.lang.String, java.lang.String\)](#)

notationDecl

```
public void notationDecl(java.lang.String name,  
                           java.lang.String publicId,  
                           java.lang.String systemId)  
    throws SAXException
```

Receive notification of a notation declaration.

By default, do nothing. Application writers may override this method in a subclass if they wish to keep track of the notations declared in a document.

Specified by:

[notationDecl](#) in interface [DTDHandler](#)

Parameters:

`name` - The notation name.

`publicId` - The notation public identifier, or null if not available.

`systemId` - The notation system identifier.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[DTDHandler.notationDecl\(java.lang.String, java.lang.String, java.lang.String\)](#)

unparsedEntityDecl

```
public void unparsedEntityDecl( java.lang.String name,
                                java.lang.String publicId,
                                java.lang.String systemId,
                                java.lang.String notationName)
                                throws SAXException
```

Receive notification of an unparsed entity declaration.

By default, do nothing. Application writers may override this method in a subclass to keep track of the unparsed entities declared in a document.

Specified by:

[unparsedEntityDecl](#) in interface [DTDHandler](#)

Parameters:

`name` - The entity name.

`publicId` - The entity public identifier, or null if not available.

`systemId` - The entity system identifier.

`notationName` - The name of the associated notation.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[DTDHandler.unparsedEntityDecl\(java.lang.String, java.lang.String, java.lang.String, java.lang.String\)](#)

setDocumentLocator

```
public void setDocumentLocator(Locator locator)
```

Receive a Locator object for document events.

By default, do nothing. Application writers may override this method in a subclass if they wish to store the locator for use with other document events.

Specified by:

[setDocumentLocator](#) in interface [ContentHandler](#)

Parameters:

`locator` - A locator for all SAX document events.

See Also:

[ContentHandler.setDocumentLocator\(org.xml.sax.Locator\)](#), [Locator](#)

startDocument

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

Receive notification of the beginning of the document.

By default, do nothing. Application writers may override this method in a subclass to take specific actions at the beginning of a document (such as allocating the root node of a tree or creating an output file).

Specified by:

[startDocument](#) in interface [ContentHandler](#)

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ContentHandler.startDocument\(\)](#)

endDocument

```
public void endDocument()  
           throws SAXException
```

Receive notification of the end of the document.

By default, do nothing. Application writers may override this method in a subclass to take specific actions at the end of a document (such as finalising a tree or closing an output file).

Specified by:

[endDocument](#) in interface [ContentHandler](#)

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ContentHandler.endDocument\(\)](#)

startPrefixMapping

```
public void startPrefixMapping(java.lang.String prefix,  
                               java.lang.String uri)  
           throws SAXException
```

Receive notification of the start of a Namespace mapping.

By default, do nothing. Application writers may override this method in a subclass to take specific actions at the start of each Namespace prefix scope (such as storing the prefix mapping).

Specified by:

[startPrefixMapping](#) in interface [ContentHandler](#)

Parameters:

`prefix` - The Namespace prefix being declared.

`uri` - The Namespace URI mapped to the prefix.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ContentHandler.startPrefixMapping\(java.lang.String, java.lang.String\)](#)

endPrefixMapping

```
public void endPrefixMapping(java.lang.String prefix)
                        throws SAXException
```

Receive notification of the end of a Namespace mapping.

By default, do nothing. Application writers may override this method in a subclass to take specific actions at the end of each prefix mapping.

Specified by:

[endPrefixMapping](#) in interface [ContentHandler](#)

Parameters:

`prefix` - The Namespace prefix being declared.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ContentHandler.endPrefixMapping\(java.lang.String\)](#)

startElement

```
public void startElement(java.lang.String uri,
                        java.lang.String localName,
                        java.lang.String qName,
```


[Attributes](#) attributes)
throws [SAXException](#)

Receive notification of the start of an element.

By default, do nothing. Application writers may override this method in a subclass to take specific actions at the start of each element (such as allocating a new tree node or writing output to a file).

Specified by:

[startElement](#) in interface [ContentHandler](#)

Parameters:

name - The element type name.

attributes - The specified or defaulted attributes.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ContentHandler.startElement\(java.lang.String,
java.lang.String, java.lang.String, org.xml.sax.Attributes\)](#)

endElement

```
public void endElement(java.lang.String uri,  
                        java.lang.String localName,  
                        java.lang.String qName)  
    throws SAXException
```

Receive notification of the end of an element.

By default, do nothing. Application writers may override this method in a subclass to take specific actions at the end of each element (such as finalising a tree node or writing output to a file).

Specified by:

[endElement](#) in interface [ContentHandler](#)

Parameters:

name - The element type name.

attributes - The specified or defaulted attributes.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ContentHandler.endElement\(java.lang.String, java.lang.String,
java.lang.String\)](#)

characters

```
public void characters(char[] ch,  
                      int start,  
                      int length)  
    throws SAXException
```

Receive notification of character data inside an element.

By default, do nothing. Application writers may override this method to take specific actions for each chunk of character data (such as adding the data to a node or buffer, or printing it to a file).

Specified by:

[characters](#) in interface [ContentHandler](#)

Parameters:

ch - The characters.

start - The start position in the character array.

length - The number of characters to use from the character array.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ContentHandler.characters\(char\[\], int, int\)](#)

ignorableWhitespace

```
public void ignorableWhitespace(char[] ch,  
                                int start,  
                                int length)  
    throws SAXException
```

Receive notification of ignorable whitespace in element content.

By default, do nothing. Application writers may override this method to take specific actions for each chunk of ignorable whitespace (such as adding data to a node or buffer, or printing it to a file).

Specified by:

[ignorableWhitespace](#) in interface [ContentHandler](#)

Parameters:

ch - The whitespace characters.

start - The start position in the character array.

length - The number of characters to use from the character array.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ContentHandler.ignorableWhitespace\(char\[\], int, int\)](#)

processingInstruction

```
public void processingInstruction(java.lang.String target,
                                   java.lang.String data)
                                   throws SAXException
```

Receive notification of a processing instruction.

By default, do nothing. Application writers may override this method in a subclass to take specific actions for each processing instruction, such as setting status variables or invoking other methods.

Specified by:

[processingInstruction](#) in interface [ContentHandler](#)

Parameters:

target - The processing instruction target.

data - The processing instruction data, or null if none is supplied.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ContentHandler.processingInstruction\(java.lang.String, java.lang.String\)](#)

skippedEntity

```
public void skippedEntity(java.lang.String name)
                           throws SAXException
```

Receive notification of a skipped entity.

By default, do nothing. Application writers may override this method in a subclass to take specific actions for each processing instruction, such as setting status variables or invoking other methods.

Specified by:

[skippedEntity](#) in interface [ContentHandler](#)

Parameters:

name - The name of the skipped entity.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ContentHandler.processingInstruction\(java.lang.String, java.lang.String\)](#)

warning

```
public void warning(SAXParseException e)
    throws SAXException
```

Receive notification of a parser warning.

The default implementation does nothing. Application writers may override this method in a subclass to take specific actions for each warning, such as inserting the message in a log file or printing it to the console.

Specified by:

[warning](#) in interface [ErrorHandler](#)

Parameters:

e - The warning information encoded as an exception.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ErrorHandler.warning\(org.xml.sax.SAXParseException\), SAXParseException](#)

error

```
public void error(SAXParseException e)
    throws SAXException
```

Receive notification of a recoverable parser error.

The default implementation does nothing. Application writers may override this method in a subclass to take specific actions for each error, such as inserting the message in a log file or printing it to the console.

Specified by:

[error](#) in interface [ErrorHandler](#)

Parameters:

e - The warning information encoded as an exception.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ErrorHandler.warning\(org.xml.sax.SAXParseException\)](#),
[SAXParseException](#)

fatalError

```
public void fatalError(SAXParseException e)
    throws SAXException
```

Report a fatal XML parsing error.

The default implementation throws a [SAXParseException](#). Application writers may override this method in a subclass if they need to take specific actions for each fatal error (such as collecting all of the errors into a single report): in any case, the application must stop all regular processing when this method is invoked, since the document is no longer reliable, and the parser may no longer report parsing events.

Specified by:

[fatalError](#) in interface [ErrorHandler](#)

Parameters:

e - The error information encoded as an exception.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ErrorHandler.fatalError\(org.xml.sax.SAXParseException\)](#),
[SAXParseException](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Interface ContentHandler

All Known Subinterfaces:

[TemplatesHandler](#), [TransformerHandler](#)

All Known Implementing Classes:

[XMLReaderAdapter](#), [XMLFilterImpl](#), [DefaultHandler](#)

public abstract interface **ContentHandler**

Receive notification of the logical content of a document.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This is the main interface that most SAX applications implement: if the application needs to be informed of basic parsing events, it implements this interface and registers an instance with the SAX parser using the [setContentHandler](#) method. The parser uses the instance to report basic document-related events like the start and end of elements and character data.

The order of events in this interface is very important, and mirrors the order of information in the document itself. For example, all of an element's content (character data, processing instructions, and/or subelements) will appear, in order, between the startElement event and the corresponding endElement event.

This interface is similar to the now-deprecated SAX 1.0 DocumentHandler interface, but it adds support for Namespaces and for reporting skipped entities (in non-validating XML processors).

Implementors should note that there is also a Java class ContentHandler in the java.net package; that means that it's probably a bad idea to do

```
import java.net.*; import org.xml.sax.*;
```

In fact, "import ...*" is usually a sign of sloppy programming anyway, so the user should consider this a feature rather than a bug.

Since:

SAX 2.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[XMLReader](#), [DTDHandler](#), [ErrorHandler](#)

Method Summary

void	characters (char[] ch, int start, int length) Receive notification of character data.
void	endDocument () Receive notification of the end of a document.
void	endElement (java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName) Receive notification of the end of an element.
void	endPrefixMapping (java.lang.String prefix) End the scope of a prefix-URI mapping.
void	ignorableWhitespace (char[] ch, int start, int length) Receive notification of ignorable whitespace in element content.
void	processingInstruction (java.lang.String target, java.lang.String data) Receive notification of a processing instruction.
void	setDocumentLocator (Locator locator) Receive an object for locating the origin of SAX document events.
void	skippedEntity (java.lang.String name) Receive notification of a skipped entity.
void	startDocument () Receive notification of the beginning of a document.
void	startElement (java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName, Attributes atts) Receive notification of the beginning of an element.
void	startPrefixMapping (java.lang.String prefix, java.lang.String uri) Begin the scope of a prefix-URI Namespace mapping.

Method Detail

setDocumentLocator

```
public void setDocumentLocator(Locator locator)
```

Receive an object for locating the origin of SAX document events.

SAX parsers are strongly encouraged (though not absolutely required) to supply a locator: if it does so, it must supply the locator to the application by invoking this method before invoking any of the other methods in the ContentHandler interface.

The locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application will use this information for reporting its own errors (such as character content that does not match an application's business rules). The information returned by the locator is probably not sufficient for use with a search engine.

Note that the locator will return correct information only during the invocation of the events in this interface. The application should not attempt to use it at any other time.

Parameters:

`locator` - An object that can return the location of any SAX document event.

See Also:

[Locator](#)

startDocument

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

Receive notification of the beginning of a document.

The SAX parser will invoke this method only once, before any other methods in this interface or in [DTDHandler](#) (except for [setDocumentLocator](#)).

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[endDocument\(\)](#)

endDocument

```
public void endDocument()  
           throws SAXException
```


Receive notification of the end of a document.

The SAX parser will invoke this method only once, and it will be the last method invoked during the parse. The parser shall not invoke this method until it has either abandoned parsing (because of an unrecoverable error) or reached the end of input.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[startDocument\(\)](#)

startPrefixMapping

```
public void startPrefixMapping(java.lang.String prefix,  
                               java.lang.String uri)  
    throws SAXException
```

Begin the scope of a prefix-URI Namespace mapping.

The information from this event is not necessary for normal Namespace processing: the SAX XML reader will automatically replace prefixes for element and attribute names when the `http://xml.org/sax/features/namespaces` feature is true (the default).

There are cases, however, when applications need to use prefixes in character data or in attribute values, where they cannot safely be expanded automatically; the start/endPrefixMapping event supplies the information to the application to expand prefixes in those contexts itself, if necessary.

Note that start/endPrefixMapping events are not guaranteed to be properly nested relative to each-other: all startPrefixMapping events will occur before the corresponding [startElement](#) event, and all [endPrefixMapping](#) events will occur after the corresponding [endElement](#) event, but their order is not otherwise guaranteed.

There should never be start/endPrefixMapping events for the "xml" prefix, since it is predeclared and immutable.

Parameters:

`prefix` - The Namespace prefix being declared.

`uri` - The Namespace URI the prefix is mapped to.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[endPrefixMapping\(java.lang.String\)](#),
[startElement\(java.lang.String, java.lang.String,
java.lang.String, org.xml.sax.Attributes\)](#)

endPrefixMapping

```
public void endPrefixMapping(java.lang.String prefix)
    throws SAXException
```

End the scope of a prefix-URI mapping.

See [startPrefixMapping](#) for details. This event will always occur after the corresponding [endElement](#) event, but the order of [endPrefixMapping](#) events is not otherwise guaranteed.

Parameters:

`prefix` - The prefix that was being mapping.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[startPrefixMapping\(java.lang.String, java.lang.String\)](#),
[endElement\(java.lang.String, java.lang.String, java.lang.String\)](#)

startElement

```
public void startElement(java.lang.String namespaceURI,
    java.lang.String localName,
    java.lang.String qName,
    Attributes atts)
    throws SAXException
```

Receive notification of the beginning of an element.

The Parser will invoke this method at the beginning of every element in the XML document; there will be a corresponding [endElement](#) event for every startElement event (even when the element is empty). All of the element's content will be reported, in order, before the corresponding endElement event.

This event allows up to three name components for each element:

1. the Namespace URI;
2. the local name; and
3. the qualified (prefixed) name.

Any or all of these may be provided, depending on the values of the <http://xml.org/sax/features/namespaces> and the <http://xml.org/sax/features/namespace-prefixes> properties:

- the Namespace URI and local name are required when the namespaces property is true (the default), and are optional when the namespaces property is false (if one is specified, both must be);
- the qualified name is required when the namespace-prefixes property is true, and is optional when the namespace-prefixes property is false (the default).

Note that the attribute list provided will contain only attributes with explicit values (specified or defaulted): #IMPLIED attributes will be omitted. The attribute list will contain attributes used for Namespace declarations (xmlns* attributes) only if the `http://xml.org/sax/features/namespace-prefixes` property is true (it is false by default, and support for a true value is optional).

Parameters:

`uri` - The Namespace URI, or the empty string if the element has no Namespace URI or if Namespace processing is not being performed.

`localName` - The local name (without prefix), or the empty string if Namespace processing is not being performed.

`qName` - The qualified name (with prefix), or the empty string if qualified names are not available.

`atts` - The attributes attached to the element. If there are no attributes, it shall be an empty Attributes object.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[endElement\(java.lang.String, java.lang.String, java.lang.String\), Attributes](#)

endElement

```
public void endElement(java.lang.String namespaceURI,
                       java.lang.String localName,
                       java.lang.String qName)
    throws SAXException
```

Receive notification of the end of an element.

The SAX parser will invoke this method at the end of every element in the XML document; there will be a corresponding [startElement](#) event for every endElement event (even when the element is empty).

For information on the names, see startElement.

Parameters:

`uri` - The Namespace URI, or the empty string if the element has no Namespace URI or if

Namespace processing is not being performed.

`localName` - The local name (without prefix), or the empty string if Namespace processing is not being performed.

`qName` - The qualified XML 1.0 name (with prefix), or the empty string if qualified names are not available.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

characters

```
public void characters(char[] ch,
                       int start,
                       int length)
    throws SAXException
```

Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity so that the Locator provides useful information.

The application must not attempt to read from the array outside of the specified range.

Note that some parsers will report whitespace in element content using the [ignorableWhitespace](#) method rather than this one (validating parsers *must* do so).

Parameters:

`ch` - The characters from the XML document.

`start` - The start position in the array.

`length` - The number of characters to read from the array.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ignorableWhitespace\(char\[\], int, int\)](#), [Locator](#)

ignorableWhitespace

```
public void ignorableWhitespace(char[] ch,
                                int start,
                                int length)
```

throws [SAXException](#)

Receive notification of ignorable whitespace in element content.

Validating Parsers must use this method to report each chunk of whitespace in element content (see the W3C XML 1.0 recommendation, section 2.10): non-validating parsers may also use this method if they are capable of parsing and using content models.

SAX parsers may return all contiguous whitespace in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

The application must not attempt to read from the array outside of the specified range.

Parameters:

`ch` - The characters from the XML document.

`start` - The start position in the array.

`length` - The number of characters to read from the array.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[characters\(char\[\], int, int\)](#)

processingInstruction

```
public void processingInstruction(java.lang.String target,  
                                   java.lang.String data)  
    throws SAXException
```

Receive notification of a processing instruction.

The Parser will invoke this method once for each processing instruction found: note that processing instructions may occur before or after the main document element.

A SAX parser must never report an XML declaration (XML 1.0, section 2.8) or a text declaration (XML 1.0, section 4.3.1) using this method.

Parameters:

`target` - The processing instruction target.

`data` - The processing instruction data, or null if none was supplied. The data does not include any whitespace separating it from the target.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

skippedEntity

```
public void skippedEntity(java.lang.String name)
    throws SAXException
```

Receive notification of a skipped entity.

The Parser will invoke this method once for each entity skipped. Non-validating processors may skip entities if they have not seen the declarations (because, for example, the entity was declared in an external DTD subset). All processors may skip external entities, depending on the values of the <http://xml.org/sax/features/external-general-entities> and the <http://xml.org/sax/features/external-parameter-entities> properties.

Parameters:

name - The name of the skipped entity. If it is a parameter entity, the name will begin with '%', and if it is the external DTD subset, it will be the string "[dtd]".

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: INNER | FIELD | CONSTR | [METHOD](#)DETAIL: FIELD | CONSTR | [METHOD](#)**org.xml.sax**

Interface DTDHandler

All Known Subinterfaces:

[TransformerHandler](#)

All Known Implementing Classes:

[HandlerBase](#), [XMLFilterImpl](#), [DefaultHandler](#)

public abstract interface **DTDHandler**

Receive notification of basic DTD-related events.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

If a SAX application needs information about notations and unparsed entities, then the application implements this interface and registers an instance with the SAX parser using the parser's `setDTDHandler` method. The parser uses the instance to report notation and unparsed entity declarations to the application.

Note that this interface includes only those DTD events that the XML recommendation *requires* processors to report: notation and unparsed entity declarations.

The SAX parser may report these events in any order, regardless of the order in which the notations and unparsed entities were declared; however, all DTD events must be reported after the document handler's `startDocument` event, and before the first `startElement` event.

It is up to the application to store the information for future use (perhaps in a hash table or object tree). If the application encounters attributes of type "NOTATION", "ENTITY", or "ENTITIES", it can use the information that it obtained through this interface to find the entity and/or notation corresponding with the attribute value.

Since:

SAX 1.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[Parser.setDTDHandler\(org.xml.sax.DTDHandler\)](#), [HandlerBase](#)

Method Summary

void	notationDecl (java.lang.String name, java.lang.String publicId, java.lang.String systemId) Receive notification of a notation declaration event.
void	unparsedEntityDecl (java.lang.String name, java.lang.String publicId, java.lang.String systemId, java.lang.String notationName) Receive notification of an unparsed entity declaration event.

Method Detail

notationDecl

```
public void notationDecl(java.lang.String name,
                          java.lang.String publicId,
                          java.lang.String systemId)
    throws SAXException
```

Receive notification of a notation declaration event.

It is up to the application to record the notation for later reference, if necessary.

At least one of publicId and systemId must be non-null. If a system identifier is present, and it is a URL, the SAX parser must resolve it fully before passing it to the application through this event.

There is no guarantee that the notation declaration will be reported before any unparsed entities that use it.

Parameters:

name - The notation name.

publicId - The notation's public identifier, or null if none was given.

systemId - The notation's system identifier, or null if none was given.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[unparsedEntityDecl\(java.lang.String, java.lang.String, java.lang.String, java.lang.String\)](#), [AttributeList](#)

unparsedEntityDecl

```
public void unparsedEntityDecl( java.lang.String name,
                                java.lang.String publicId,
                                java.lang.String systemId,
                                java.lang.String notationName)
    throws SAXException
```

Receive notification of an unparsed entity declaration event.

Note that the notation name corresponds to a notation reported by the [notationDecl](#) event. It is up to the application to record the entity for later reference, if necessary.

If the system identifier is a URL, the parser must resolve it fully before passing it to the application.

Parameters:

`name` - The unparsed entity's name.

`publicId` - The entity's public identifier, or null if none was given.

`systemId` - The entity's system identifier.

`notation` - name The name of the associated notation.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[notationDecl\(java.lang.String, java.lang.String, java.lang.String\), AttributeList](#)

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)**org.xml.sax**

Interface EntityResolver

All Known Implementing Classes:

[HandlerBase](#), [XMLFilterImpl](#), [DefaultHandler](#)
public abstract interface **EntityResolver**

Basic interface for resolving entities.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

If a SAX application needs to implement customized handling for external entities, it must implement this interface and register an instance with the SAX driver using the [setEntityResolver](#) method.

The XML reader will then allow the application to intercept any external entities (including the external DTD subset and external parameter entities, if any) before including them.

Many SAX applications will not need to implement this interface, but it will be especially useful for applications that build XML documents from databases or other specialised input sources, or for applications that use URI types other than URLs.

The following resolver would provide the application with a special character stream for the entity with the system identifier "http://www.myhost.com/today":

```
import org.xml.sax.EntityResolver;
import org.xml.sax.InputSource;

public class MyResolver implements EntityResolver {
    public InputSource resolveEntity (String publicId, String systemId)
    {
        if (systemId.equals("http://www.myhost.com/today")) {
            // return a special input source
            MyReader reader = new MyReader();
            return new InputSource(reader);
        } else {
            // use the default behaviour
            return null;
        }
    }
}
```

The application can also use this interface to redirect system identifiers to local URIs or to look up replacements in a catalog (possibly by using the public identifier).

Since:

SAX 1.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[Parser.setEntityResolver\(org.xml.sax.EntityResolver\)](#), [InputSource](#)

Method Summary

InputSource	resolveEntity (java.lang.String publicId, java.lang.String systemId) Allow the application to resolve external entities.
-----------------------------	--

Method Detail

resolveEntity

```
public InputSource resolveEntity( java.lang.String publicId,  

                                   java.lang.String systemId)  

    throws SAXException,  

           java.io.IOException
```

Allow the application to resolve external entities.

The Parser will call this method before opening any external entity except the top-level document entity (including the external DTD subset, external entities referenced within the DTD, and external entities referenced within the document element): the application may request that the parser resolve the entity itself, that it use an alternative URI, or that it use an entirely different input source.

Application writers can use this method to redirect external system identifiers to secure and/or local URIs, to look up public identifiers in a catalogue, or to read an entity from a database or other input source (including, for example, a dialog box).

If the system identifier is a URL, the SAX parser must resolve it fully before reporting it to the application.

Parameters:

`publicId` - The public identifier of the external entity being referenced, or null if none was supplied.

`systemId` - The system identifier of the external entity being referenced.

Returns:

An `InputSource` object describing the new input source, or null to request that the parser open a regular URI connection to the system identifier.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

`java.io.IOException` - A Java-specific IO exception, possibly the result of creating a new `InputStream` or `Reader` for the `InputSource`.

See Also:

[InputSource](#)

[Overview](#) [Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Interface ErrorHandler

All Known Implementing Classes:

[HandlerBase](#), [XMLFilterImpl](#), [DefaultHandler](#)

public abstract interface **ErrorHandler**

Basic interface for SAX error handlers.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

If a SAX application needs to implement customized error handling, it must implement this interface and then register an instance with the XML reader using the [setErrorHandler](#) method. The parser will then report all errors and warnings through this interface.

WARNING: If an application does *not* register an ErrorHandler, XML parsing errors will go unreported and bizarre behaviour may result.

For XML processing errors, a SAX driver must use this interface instead of throwing an exception: it is up to the application to decide whether to throw an exception for different types of errors and warnings. Note, however, that there is no requirement that the parser continue to provide useful information after a call to [fatalError](#) (in other words, a SAX driver class could catch an exception and report a fatalError).

Since:

SAX 1.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[Parser.setErrorHandler\(org.xml.sax.ErrorHandler\)](#), [SAXParseException](#)

Method Summary

void	error (SAXParseException exception) Receive notification of a recoverable error.
------	---

void	fatalError (SAXParseException exception) Receive notification of a non-recoverable error.
void	warning (SAXParseException exception) Receive notification of a warning.

Method Detail

warning

```
public void warning(SAXParseException exception)
    throws SAXException
```

Receive notification of a warning.

SAX parsers will use this method to report conditions that are not errors or fatal errors as defined by the XML 1.0 recommendation. The default behaviour is to take no action.

The SAX parser must continue to provide normal parsing events after invoking this method: it should still be possible for the application to process the document through to the end.

Filters may use this method to report other, non-XML warnings as well.

Parameters:

`exception` - The warning information encapsulated in a SAX parse exception.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[SAXParseException](#)

error

```
public void error(SAXParseException exception)
    throws SAXException
```

Receive notification of a recoverable error.

This corresponds to the definition of "error" in section 1.2 of the W3C XML 1.0 Recommendation. For example, a validating parser would use this callback to report the violation of a validity constraint. The default behaviour is to take no action.

The SAX parser must continue to provide normal parsing events after invoking this method: it should still be possible for the application to process the document through to the end. If the

application cannot do so, then the parser should report a fatal error even if the XML 1.0 recommendation does not require it to do so.

Filters may use this method to report other, non-XML errors as well.

Parameters:

`exception` - The error information encapsulated in a SAX parse exception.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[SAXParseException](#)

fatalError

```
public void fatalError(SAXParseException exception)
    throws SAXException
```

Receive notification of a non-recoverable error.

This corresponds to the definition of "fatal error" in section 1.2 of the W3C XML 1.0 Recommendation. For example, a parser would use this callback to report the violation of a well-formedness constraint.

The application must assume that the document is unusable after the parser has invoked this method, and should continue (if at all) only for the sake of collecting addition error messages: in fact, SAX parsers are free to stop reporting any other events once this method has been invoked.

Parameters:

`exception` - The error information encapsulated in a SAX parse exception.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[SAXParseException](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.parsers

Class DocumentBuilder

java.lang.Object

```

|
+-- javax.xml.parsers.DocumentBuilder

```

public abstract class **DocumentBuilder**

extends java.lang.Object

Defines the API to obtain DOM Document instances from an XML document. Using this class, an application programmer can obtain a [Document](#) from XML.

An instance of this class can be obtained from the [DocumentBuilderFactory.newDocumentBuilder](#) method. Once an instance of this class is obtained, XML can be parsed from a variety of input sources. These input sources are InputStreams, Files, URLs, and SAX InputSources.

Note that this class reuses several classes from the SAX API. This does not require that the implementor of the underlying DOM implementation use a SAX parser to parse XML document into a Document. It merely requires that the implementation communicate with the application using these existing APIs.

An implementation of DocumentBuilder is *NOT* guaranteed to behave as per the specification if it is used concurrently by two or more threads. It is recommended to have one instance of the DocumentBuilder per thread or it is upto the application to make sure about the use of DocumentBuilder from more than one thread.

Since:

JAXP 1.0

Version:

1.0

Constructor Summary

protected	DocumentBuilder ()
-----------	-------------------------------------

Method Summary

abstract DOMImplementation	getDOMImplementation () Obtain an instance of a DOMImplementation object.
abstract boolean	isNamespaceAware () Indicates whether or not this parser is configured to understand namespaces.
abstract boolean	isValidating () Indicates whether or not this parser is configured to validate XML documents.
abstract Document	newDocument () Obtain a new instance of a DOM Document object to build a DOM tree with.
Document	parse (java.io.File f) Parse the content of the given file as an XML document and return a new DOM Document object.
abstract Document	parse (InputSource is) Parse the content of the given input source as an XML document and return a new DOM Document object.
Document	parse (java.io.InputStream is) Parse the content of the given InputStream as an XML document and return a new DOM Document object.
Document	parse (java.io.InputStream is, java.lang.String systemId) Parse the content of the given InputStream as an XML document and return a new DOM Document object.
Document	parse (java.lang.String uri) Parse the content of the given URI as an XML document and return a new DOM Document object.
abstract void	setEntityResolver (EntityResolver er) Specify the EntityResolver to be used to resolve entities present in the XML document to be parsed.
abstract void	setErrorHandler (ErrorHandler eh) Specify the ErrorHandler to be used to report errors present in the XML document to be parsed.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail**DocumentBuilder**

protected **DocumentBuilder**()

Method Detail**parse**

```
public Document parse(java.io.InputStream is)
    throws SAXException,
           java.io.IOException
```

Parse the content of the given InputStream as an XML document and return a new DOM [Document](#) object.

Parameters:

`is` - InputStream containing the content to be parsed.

Throws:

java.io.IOException - If any IO errors occur.

[SAXException](#) - If any parse errors occur.

java.lang.IllegalArgumentException - If the InputStream is null

See Also:

[DocumentHandler](#)

parse

```
public Document parse(java.io.InputStream is,
    java.lang.String systemId)
    throws SAXException,
           java.io.IOException
```

Parse the content of the given InputStream as an XML document and return a new DOM

[Document](#) object.

Parameters:

`is` - InputStream containing the content to be parsed.
`systemId` - Provide a base for resolving relative URIs.

Returns:

A new DOM Document object.

Throws:

`java.io.IOException` - If any IO errors occur.
[SAXException](#) - If any parse errors occur.
`java.lang.IllegalArgumentException` - If the InputStream is null.

See Also:

[DocumentHandler](#)

parse

```
public Document parse(java.lang.String uri)
    throws SAXException,
           java.io.IOException
```

Parse the content of the given URI as an XML document and return a new DOM [Document](#) object.

Parameters:

`uri` - The location of the content to be parsed.

Returns:

A new DOM Document object.

Throws:

`java.io.IOException` - If any IO errors occur.
[SAXException](#) - If any parse errors occur.
`java.lang.IllegalArgumentException` - If the URI is null.

See Also:

[DocumentHandler](#)

parse

```
public Document parse(java.io.File f)
    throws SAXException,
```

`java.io.IOException`

Parse the content of the given file as an XML document and return a new DOM [Document](#) object.

Parameters:

`f` - The file containing the XML to parse.

Returns:

A new DOM Document object.

Throws:

`java.io.IOException` - If any IO errors occur.

[SAXException](#) - If any parse errors occur.

`java.lang.IllegalArgumentException` - If the file is null.

See Also:

[DocumentHandler](#)

parse

```
public abstract Document parse(InputSource is)
                               throws SAXException,
                                       java.io.IOException
```

Parse the content of the given input source as an XML document and return a new DOM [Document](#) object.

Parameters:

`is` - `InputSource` containing the content to be parsed.

Returns:

A new DOM Document object.

Throws:

`java.io.IOException` - If any IO errors occur.

[SAXException](#) - If any parse errors occur.

`java.lang.IllegalArgumentException` - If the `InputSource` is null.

See Also:

[DocumentHandler](#)

isNamespaceAware

```
public abstract boolean isNamespaceAware()
```

Indicates whether or not this parser is configured to understand namespaces.

Returns:

true if this parser is configured to understand namespaces; false otherwise.

isValidating

```
public abstract boolean isValidating()
```

Indicates whether or not this parser is configured to validate XML documents.

Returns:

true if this parser is configured to validate XML documents; false otherwise.

setEntityResolver

```
public abstract void setEntityResolver(EntityResolver er)
```

Specify the [EntityResolver](#) to be used to resolve entities present in the XML document to be parsed. Setting this to null will result in the underlying implementation using it's own default implementation and behavior.

Parameters:

er - The `EntityResolver` to be used to resolve entities present in the XML document to be parsed.

setErrorHandler

```
public abstract void setErrorHandler(ErrorHandler eh)
```

Specify the [ErrorHandler](#) to be used to report errors present in the XML document to be parsed. Setting this to null will result in the underlying implementation using it's own default implementation and behavior.

Parameters:

eh - The `ErrorHandler` to be used to report errors present in the XML document to be parsed.

newDocument

```
public abstract Document newDocument()
```

Obtain a new instance of a DOM [Document](#) object to build a DOM tree with. An alternative way to create a DOM Document object is to use the [getDOMImplementation](#) method to get a DOM Level 2 DOMImplementation object and then use DOM Level 2 methods on that object to create a DOM Document object.

Returns:

A new instance of a DOM Document object.

getDOMImplementation

```
public abstract DOMImplementation getDOMImplementation( )
```

Obtain an instance of a [DOMImplementation](#) object.

Returns:

A new instance of a DOMImplementation.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.parsers

Class DocumentBuilderFactory

java.lang.Object

|
+-- **javax.xml.parsers.DocumentBuilderFactory**public abstract class **DocumentBuilderFactory**

extends java.lang.Object

Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents. An implementation of the `DocumentBuilderFactory` class is *NOT* guaranteed to be thread safe. It is up to the user application to make sure about the use of the `DocumentBuilderFactory` from more than one thread. Alternatively the application can have one instance of the `DocumentBuilderFactory` per thread. An application can use the same instance of the factory to obtain one or more instances of the `DocumentBuilder` provided the instance of the factory isn't being used in more than one thread at a time.

Since:

JAXP 1.0

Version:

1.0

Constructor Summary

protected [DocumentBuilderFactory](#)()

Method Summary

abstract java.lang.Object [getAttribute](#)(java.lang.String name)

Allows the user to retrieve specific attributes on the underlying implementation.

boolean [isCoalescing](#)()

Indicates whether or not the factory is configured to produce parsers which converts CDATA nodes to Text nodes and appends it to the adjacent (if any) Text node.

boolean [isExpandEntityReferences](#)()

Indicates whether or not the factory is configured to produce parsers which expand entity reference nodes.

boolean [isIgnoringComments](#)()

Indicates whether or not the factory is configured to produce parsers which ignores comments.

boolean	<code>isIgnoringElementContentWhitespace</code> () Indicates whether or not the factory is configured to produce parsers which ignore ignorable whitespace in element content.
boolean	<code>isNamespaceAware</code> () Indicates whether or not the factory is configured to produce parsers which are namespace aware.
boolean	<code>isValidating</code> () Indicates whether or not the factory is configured to produce parsers which validate the XML content during parse.
abstract DocumentBuilder	<code>newDocumentBuilder</code> () Creates a new instance of a DocumentBuilder using the currently configured parameters.
static DocumentBuilderFactory	<code>newInstance</code> () Obtain a new instance of a DocumentBuilderFactory.
abstract void	<code>setAttribute</code> (java.lang.String name, java.lang.Object value) Allows the user to set specific attributes on the underlying implementation.
void	<code>setCoalescing</code> (boolean coalescing) Specifies that the parser produced by this code will convert CDATA nodes to Text nodes and append it to the adjacent (if any) text node.
void	<code>setExpandEntityReferences</code> (boolean expandEntityRef) Specifies that the parser produced by this code will expand entity reference nodes.
void	<code>setIgnoringComments</code> (boolean ignoreComments) Specifies that the parser produced by this code will ignore comments.
void	<code>setIgnoringElementContentWhitespace</code> (boolean whitespace) Specifies that the parsers created by this factory must eliminate whitespace in element content (sometimes known loosely as 'ignorable whitespace') when parsing XML documents (see XML Rec 2.10).
void	<code>setNamespaceAware</code> (boolean awareness) Specifies that the parser produced by this code will provide support for XML namespaces.
void	<code>setValidating</code> (boolean validating) Specifies that the parser produced by this code will validate documents as they are parsed.

Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`,
`toString`, `wait`, `wait`, `wait`

Constructor Detail

DocumentBuilderFactory

protected **DocumentBuilderFactory**()

Method Detail

newInstance

public static [DocumentBuilderFactory](#) **newInstance**()
throws [FactoryConfigurationError](#)

Obtain a new instance of a DocumentBuilderFactory. This static method creates a new factory instance. This method uses the following ordered lookup procedure to determine the DocumentBuilderFactory implementation class to load:

- Use the `javax.xml.parsers.DocumentBuilderFactory` system property.
- Use the properties file "lib/jaxp.properties" in the JRE directory. This configuration file is in standard `java.util.Properties` format and contains the fully qualified name of the implementation class with the key being the system property defined above.
- Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file `META-INF/services/javax.xml.parsers.DocumentBuilderFactory` in jars available to the runtime.
- Platform default DocumentBuilderFactory instance.

Once an application has obtained a reference to a DocumentBuilderFactory it can use the factory to configure and obtain parser instances.

Throws:

[FactoryConfigurationError](#) - if the implementation is not available or cannot be instantiated.

newDocumentBuilder

public abstract [DocumentBuilder](#) **newDocumentBuilder**()
throws [ParserConfigurationException](#)

Creates a new instance of a [DocumentBuilder](#) using the currently configured parameters.

Returns:

A new instance of a DocumentBuilder.

Throws:

[ParserConfigurationException](#) - if a DocumentBuilder cannot be created which satisfies the configuration requested.

setNamespaceAware

public void **setNamespaceAware**(boolean awareness)

Specifies that the parser produced by this code will provide support for XML namespaces. By default the value of

this is set to false

Parameters:

awareness - true if the parser produced will provide support for XML namespaces; false otherwise.

setValidating

```
public void setValidating(boolean validating)
```

Specifies that the parser produced by this code will validate documents as they are parsed. By default the value of this is set to false.

Parameters:

validating - true if the parser produced will validate documents as they are parsed; false otherwise.

setIgnoringElementContentWhitespace

```
public void setIgnoringElementContentWhitespace(boolean whitespace)
```

Specifies that the parsers created by this factory must eliminate whitespace in element content (sometimes known loosely as 'ignorable whitespace') when parsing XML documents (see XML Rec 2.10). Note that only whitespace which is directly contained within element content that has an element only content model (see XML Rec 3.2.1) will be eliminated. Due to reliance on the content model this setting requires the parser to be in validating mode. By default the value of this is set to false.

Parameters:

whitespace - true if the parser created must eliminate whitespace in the element content when parsing XML documents; false otherwise.

setExpandEntityReferences

```
public void setExpandEntityReferences(boolean expandEntityRef)
```

Specifies that the parser produced by this code will expand entity reference nodes. By default the value of this is set to true

Parameters:

expandEntityRef - true if the parser produced will expand entity reference nodes; false otherwise.

setIgnoringComments

```
public void setIgnoringComments(boolean ignoreComments)
```

Specifies that the parser produced by this code will ignore comments. By default the value of this is set to false

setCoalescing

```
public void setCoalescing(boolean coalescing)
```

Specifies that the parser produced by this code will convert CDATA nodes to Text nodes and append it to the adjacent (if any) text node. By default the value of this is set to `false`

Parameters:

`coalescing` - true if the parser produced will convert CDATA nodes to Text nodes and append it to the adjacent (if any) text node; false otherwise.

isNamespaceAware

```
public boolean isNamespaceAware()
```

Indicates whether or not the factory is configured to produce parsers which are namespace aware.

Returns:

true if the factory is configured to produce parsers which are namespace aware; false otherwise.

isValidating

```
public boolean isValidating()
```

Indicates whether or not the factory is configured to produce parsers which validate the XML content during parse.

Returns:

true if the factory is configured to produce parsers which validate the XML content during parse; false otherwise.

isIgnoringElementContentWhitespace

```
public boolean isIgnoringElementContentWhitespace()
```

Indicates whether or not the factory is configured to produce parsers which ignore ignorable whitespace in element content.

Returns:

true if the factory is configured to produce parsers which ignore ignorable whitespace in element content; false otherwise.

isExpandEntityReferences

```
public boolean isExpandEntityReferences()
```

Indicates whether or not the factory is configured to produce parsers which expand entity reference nodes.

Returns:

true if the factory is configured to produce parsers which expand entity reference nodes; false otherwise.

isIgnoringComments

```
public boolean isIgnoringComments()
```

Indicates whether or not the factory is configured to produce parsers which ignores comments.

Returns:

true if the factory is configured to produce parsers which ignores comments; false otherwise.

isCoalescing

```
public boolean isCoalescing()
```

Indicates whether or not the factory is configured to produce parsers which converts CDATA nodes to Text nodes and appends it to the adjacent (if any) Text node.

Returns:

true if the factory is configured to produce parsers which converts CDATA nodes to Text nodes and appends it to the adjacent (if any) Text node; false otherwise.

setAttribute

```
public abstract void setAttribute(java.lang.String name,  
                                   java.lang.Object value)  
                                   throws java.lang.IllegalArgumentException
```

Allows the user to set specific attributes on the underlying implementation.

Parameters:

name - The name of the attribute.

value - The value of the attribute.

Throws:

java.lang.IllegalArgumentException - thrown if the underlying implementation doesn't recognize the attribute.

getAttribute

```
public abstract java.lang.Object getAttribute(java.lang.String name)  
                                           throws java.lang.IllegalArgumentException
```

Allows the user to retrieve specific attributes on the underlying implementation.

Parameters:

name - The name of the attribute.

Returns:

value The value of the attribute.

Throws:

java.lang.IllegalArgumentException - thrown if the underlying implementation doesn't recognize the attribute.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform.dom

Class DOMResult

java.lang.Object

```

|
+-- javax.xml.transform.dom.DOMResult

```

public class **DOMResult**

extends java.lang.Object

implements [Result](#)

Acts as a holder for a transformation result tree, in the form of a Document Object Model (DOM) tree. If no output DOM source is set, the transformation will create a Document node as the holder for the result of the transformation, which may be retrieved with getNode.

Field Summary

static java.lang.String	FEATURE If TransformerFactory.getFeature(java.lang.String) returns true when passed this value as an argument, the Transformer supports Result output of this type.
-------------------------	--

Constructor Summary

DOMResult ()	Zero-argument default constructor.
DOMResult (Node node)	Use a DOM node to create a new output target.
DOMResult (Node node, java.lang.String systemID)	Create a new output target with a DOM node.

Method Summary

Node	getNode () Get the node that will contain the result DOM tree.
java.lang.String	getSystemId () Get the system identifier that was set with setSystemId.
void	setNode (Node node) Set the node that will contain the result DOM tree.
void	setSystemId (java.lang.String systemId) Method setSystemId Set the systemID that may be used in association with the node.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

FEATURE

```
public static final java.lang.String FEATURE
```

If [TransformerFactory.getFeature\(java.lang.String\)](#) returns true when passed this value as an argument, the Transformer supports Result output of this type.

Constructor Detail

DOMResult

```
public DOMResult()
```

Zero-argument default constructor.

DOMResult

```
public DOMResult(Node node)
```

Use a DOM node to create a new output target. In practice, the node should be a [Document](#) node, a [DocumentFragment](#) node, or a [Element](#) node. In other words, a node that accepts children.

Parameters:

n - The DOM node that will contain the result tree.

DOMResult

```
public DOMResult(Node node,  
                java.lang.String systemID)
```

Create a new output target with a DOM node. In practice, the node should be a [Document](#) node, a [DocumentFragment](#) node, or a [Element](#) node. In other words, a node that accepts children.

Parameters:

node - The DOM node that will contain the result tree.

systemID - The system identifier which may be used in association with this node.

Method Detail

setNode

```
public void setNode(Node node)
```

Set the node that will contain the result DOM tree. In practice, the node should be a [Document](#) node, a [DocumentFragment](#) node, or a [Element](#) node. In other words, a node that accepts children.

Parameters:

node - The node to which the transformation will be appended.

getNode

```
public Node getNode()
```

Get the node that will contain the result DOM tree. If no node was set via setNode, the node will be set by the transformation, and may be obtained from this method once the transformation is complete.

Returns:

The node to which the transformation will be appended.

setSystemId

```
public void setSystemId(java.lang.String systemId)
```

Method setSystemId Set the systemID that may be used in association with the node.

Specified by:

[setSystemId](#) in interface [Result](#)

Parameters:

systemId - The system identifier as a URI string.

getSystemId

```
public java.lang.String getSystemId()
```

Get the system identifier that was set with setSystemId.

Specified by:

[getSystemId](#) in interface [Result](#)

Returns:

The system identifier that was set with setSystemId, or null if setSystemId was not called.

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform

Interface Result

All Known Implementing Classes:

[DOMResult](#), [SAXResult](#), [StreamResult](#)public abstract interface **Result**

An object that implements this interface contains the information needed to build a transformation result tree.

Field Summary

static java.lang.String	PI_DISABLE_OUTPUT_ESCAPING The name of the processing instruction that is sent if the result tree disables output escaping.
static java.lang.String	PI_ENABLE_OUTPUT_ESCAPING The name of the processing instruction that is sent if the result tree enables output escaping at some point after having received a PI_DISABLE_OUTPUT_ESCAPING processing instruction.

Method Summary

java.lang.String	getSystemId () Get the system identifier that was set with setSystemId.
void	setSystemId (java.lang.String systemId) Set the system identifier for this Result.

Field Detail

PI_DISABLE_OUTPUT_ESCAPING

public static final java.lang.String **PI_DISABLE_OUTPUT_ESCAPING**

The name of the processing instruction that is sent if the result tree disables output escaping.

Normally, result tree serialization escapes & and < (and possibly other characters) when outputting text nodes. This ensures that the output is well-formed XML. However, it is sometimes convenient to be able to produce output that is almost, but not quite well-formed XML; for example, the output may include ill-formed sections that will be transformed into well-formed XML by a subsequent non-XML aware process. If a processing instruction is sent with this name, serialization should be output without any escaping.

Result DOM trees may also have `PI_DISABLE_OUTPUT_ESCAPING` and `PI_ENABLE_OUTPUT_ESCAPING` inserted into the tree.

See Also:

[disable-output-escaping in XSLT Specification](#)

PI_ENABLE_OUTPUT_ESCAPING

```
public static final java.lang.String PI_ENABLE_OUTPUT_ESCAPING
```

The name of the processing instruction that is sent if the result tree enables output escaping at some point after having received a `PI_DISABLE_OUTPUT_ESCAPING` processing instruction.

See Also:

[disable-output-escaping in XSLT Specification](#)

Method Detail

setSystemId

```
public void setSystemId(java.lang.String systemId)
```

Set the system identifier for this Result.

If the Result is not to be written to a file, the system identifier is optional. The application may still want to provide one, however, for use in error messages and warnings, or to resolve relative output identifiers.

Parameters:

`systemId` - The system identifier as a URI string.

getSystemId

```
public java.lang.String getSystemId()
```

Get the system identifier that was set with `setSystemId`.

Returns:

The system identifier that was set with `setSystemId`, or null if `setSystemId` was not called.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform

Interface Source

All Known Implementing Classes:

[DOMSource](#), [SAXSource](#), [StreamSource](#)public abstract interface **Source**

An object that implements this interface contains the information needed to act as source input (XML source or transformation instructions).

Method Summary

java.lang.String	getSystemId () Get the system identifier that was set with setSystemId.
void	setSystemId (java.lang.String systemId) Set the system identifier for this Source.

Method Detail

setSystemId

```
public void setSystemId(java.lang.String systemId)
```

Set the system identifier for this Source.

The system identifier is optional if the source does not get its data from a URL, but it may still be useful to provide one. The application can use a system identifier, for example, to resolve relative URIs and to include in error messages and warnings.

Parameters:

systemId - The system identifier as a URL string.

getSystemId

```
public java.lang.String getSystemId()
```

Get the system identifier that was set with `setSystemId`.

Returns:

The system identifier that was set with `setSystemId`, or null if `setSystemId` was not called.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Class HandlerBase

java.lang.Object

|

+--org.xml.sax.HandlerBase

Deprecated. *This class works with the deprecated [DocumentHandler](#) interface. It has been replaced by the SAX2 [DefaultHandler](#) class.*

public class **HandlerBase**

extends java.lang.Object

implements [EntityResolver](#), [DTDHandler](#), [DocumentHandler](#), [ErrorHandler](#)

Default base class for handlers.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This class implements the default behaviour for four SAX1 interfaces: EntityResolver, DTDHandler, DocumentHandler, and ErrorHandler. It is now obsolete, but is included in SAX2 to support legacy SAX1 applications. SAX2 applications should use the [DefaultHandler](#) class instead.

Application writers can extend this class when they need to implement only part of an interface; parser writers can instantiate this class to provide default handlers when the application has not supplied its own.

Note that the use of this class is optional.

Since:

SAX 1.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[EntityResolver](#), [DTDHandler](#), [DocumentHandler](#), [ErrorHandler](#)

Constructor Summary

[HandlerBase](#)()
Deprecated.

Method Summary

void	characters (char[] ch, int start, int length) Deprecated. Receive notification of character data inside an element.
void	endDocument () Deprecated. Receive notification of the end of the document.
void	endElement (java.lang.String name) Deprecated. Receive notification of the end of an element.
void	error (SAXParseException e) Deprecated. Receive notification of a recoverable parser error.
void	fatalError (SAXParseException e) Deprecated. Report a fatal XML parsing error.
void	ignorableWhitespace (char[] ch, int start, int length) Deprecated. Receive notification of ignorable whitespace in element content.
void	notationDecl (java.lang.String name, java.lang.String publicId, java.lang.String systemId) Deprecated. Receive notification of a notation declaration.
void	processingInstruction (java.lang.String target, java.lang.String data) Deprecated. Receive notification of a processing instruction.
InputSource	resolveEntity (java.lang.String publicId, java.lang.String systemId) Deprecated. Resolve an external entity.
void	setDocumentLocator (Locator locator) Deprecated. Receive a Locator object for document events.
void	startDocument () Deprecated. Receive notification of the beginning of the document.
void	startElement (java.lang.String name, AttributeList attributes) Deprecated. Receive notification of the start of an element.

void	unparsedEntityDecl (java.lang.String name, java.lang.String publicId, java.lang.String systemId, java.lang.String notationName) Deprecated. Receive notification of an unparsed entity declaration.
void	warning (SAXParseException e) Deprecated. Receive notification of a parser warning.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

HandlerBase

```
public HandlerBase()
    Deprecated.
```

Method Detail

resolveEntity

```
public InputSource resolveEntity( java.lang.String publicId,
                                   java.lang.String systemId)
    throws SAXException
```

Deprecated.

Resolve an external entity.

Always return null, so that the parser will use the system identifier provided in the XML document. This method implements the SAX default behaviour: application writers can override it in a subclass to do special translations such as catalog lookups or URI redirection.

Specified by:

[resolveEntity](#) in interface [EntityResolver](#)

Parameters:

publicId - The public identifier, or null if none is available.

systemId - The system identifier provided in the XML document.

Returns:

The new input source, or null to require the default behaviour.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[EntityResolver.resolveEntity\(java.lang.String,
java.lang.String\)](#)

notationDecl

```
public void notationDecl(java.lang.String name,  
                           java.lang.String publicId,  
                           java.lang.String systemId)
```

Deprecated.

Receive notification of a notation declaration.

By default, do nothing. Application writers may override this method in a subclass if they wish to keep track of the notations declared in a document.

Specified by:

[notationDecl](#) in interface [DTDHandler](#)

Parameters:

name - The notation name.

publicId - The notation public identifier, or null if not available.

systemId - The notation system identifier.

See Also:

[DTDHandler.notationDecl\(java.lang.String, java.lang.String,
java.lang.String\)](#)

unparsedEntityDecl

```
public void unparsedEntityDecl(java.lang.String name,  
                                java.lang.String publicId,  
                                java.lang.String systemId,  
                                java.lang.String notationName)
```

Deprecated.

Receive notification of an unparsed entity declaration.

By default, do nothing. Application writers may override this method in a subclass to keep track of the unparsed entities declared in a document.

Specified by:

[unparsedEntityDecl](#) in interface [DTDHandler](#)

Parameters:

`name` - The entity name.

`publicId` - The entity public identifier, or null if not available.

`systemId` - The entity system identifier.

`notationName` - The name of the associated notation.

See Also:

[DTDHandler.unparsedEntityDecl\(java.lang.String, java.lang.String, java.lang.String, java.lang.String\)](#)

setDocumentLocator

```
public void setDocumentLocator(Locator locator)
```

Deprecated.

Receive a Locator object for document events.

By default, do nothing. Application writers may override this method in a subclass if they wish to store the locator for use with other document events.

Specified by:

[setDocumentLocator](#) in interface [DocumentHandler](#)

Parameters:

`locator` - A locator for all SAX document events.

See Also:

[DocumentHandler.setDocumentLocator\(org.xml.sax.Locator\),
\[Locator\]\(#\)](#)

startDocument

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

Deprecated.

Receive notification of the beginning of the document.

By default, do nothing. Application writers may override this method in a subclass to take specific

actions at the beginning of a document (such as allocating the root node of a tree or creating an output file).

Specified by:

[startDocument](#) in interface [DocumentHandler](#)

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[DocumentHandler.startDocument\(\)](#)

endDocument

```
public void endDocument()  
           throws SAXException
```

Deprecated.

Receive notification of the end of the document.

By default, do nothing. Application writers may override this method in a subclass to take specific actions at the beginning of a document (such as finalising a tree or closing an output file).

Specified by:

[endDocument](#) in interface [DocumentHandler](#)

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[DocumentHandler.endDocument\(\)](#)

startElement

```
public void startElement(java.lang.String name,  
                        AttributeList attributes)  
           throws SAXException
```

Deprecated.

Receive notification of the start of an element.

By default, do nothing. Application writers may override this method in a subclass to take specific actions at the start of each element (such as allocating a new tree node or writing output to a file).

Specified by:

[startElement](#) in interface [DocumentHandler](#)

Parameters:

`name` - The element type name.

`attributes` - The specified or defaulted attributes.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[DocumentHandler.startElement\(java.lang.String, org.xml.sax.AttributeList\)](#)

endElement

```
public void endElement(java.lang.String name)
                throws SAXException
```

Deprecated.

Receive notification of the end of an element.

By default, do nothing. Application writers may override this method in a subclass to take specific actions at the end of each element (such as finalising a tree node or writing output to a file).

Specified by:

[endElement](#) in interface [DocumentHandler](#)

Parameters:

`name` - The element type name.

`attributes` - The specified or defaulted attributes.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[DocumentHandler.endElement\(java.lang.String\)](#)

characters

```
public void characters(char[] ch,
                        int start,
                        int length)
                throws SAXException
```

Deprecated.

Receive notification of character data inside an element.

By default, do nothing. Application writers may override this method to take specific actions for each chunk of character data (such as adding the data to a node or buffer, or printing it to a file).

Specified by:

[characters](#) in interface [DocumentHandler](#)

Parameters:

ch - The characters.

start - The start position in the character array.

length - The number of characters to use from the character array.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[DocumentHandler.characters\(char\[\], int, int\)](#)

ignorableWhitespace

```
public void ignorableWhitespace(char[] ch,  
                                int start,  
                                int length)  
    throws SAXException
```

Deprecated.

Receive notification of ignorable whitespace in element content.

By default, do nothing. Application writers may override this method to take specific actions for each chunk of ignorable whitespace (such as adding data to a node or buffer, or printing it to a file).

Specified by:

[ignorableWhitespace](#) in interface [DocumentHandler](#)

Parameters:

ch - The whitespace characters.

start - The start position in the character array.

length - The number of characters to use from the character array.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[DocumentHandler.ignorableWhitespace\(char\[\], int, int\)](#)

processingInstruction

```
public void processingInstruction(java.lang.String target,
                                   java.lang.String data)
                                   throws SAXException
```

Deprecated.

Receive notification of a processing instruction.

By default, do nothing. Application writers may override this method in a subclass to take specific actions for each processing instruction, such as setting status variables or invoking other methods.

Specified by:

[processingInstruction](#) in interface [DocumentHandler](#)

Parameters:

target - The processing instruction target.

data - The processing instruction data, or null if none is supplied.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[DocumentHandler.processingInstruction\(java.lang.String, java.lang.String\)](#)

warning

```
public void warning(SAXParseException e)
                   throws SAXException
```

Deprecated.

Receive notification of a parser warning.

The default implementation does nothing. Application writers may override this method in a subclass to take specific actions for each warning, such as inserting the message in a log file or printing it to the console.

Specified by:

[warning](#) in interface [ErrorHandler](#)

Parameters:

e - The warning information encoded as an exception.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ErrorHandler.warning\(org.xml.sax.SAXParseException\), SAXParseException](#)

error

```
public void error(SAXParseException e)
    throws SAXException
```

Deprecated.

Receive notification of a recoverable parser error.

The default implementation does nothing. Application writers may override this method in a subclass to take specific actions for each error, such as inserting the message in a log file or printing it to the console.

Specified by:

[error](#) in interface [ErrorHandler](#)

Parameters:

e - The warning information encoded as an exception.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ErrorHandler.warning\(org.xml.sax.SAXParseException\), SAXParseException](#)

fatalError

```
public void fatalError(SAXParseException e)
    throws SAXException
```

Deprecated.

Report a fatal XML parsing error.

The default implementation throws a `SAXParseException`. Application writers may override this method in a subclass if they need to take specific actions for each fatal error (such as collecting all of the errors into a single report): in any case, the application must stop all regular processing when this method is invoked, since the document is no longer reliable, and the parser may no longer report parsing events.

Specified by:

[fatalError](#) in interface [ErrorHandler](#)

Parameters:

e - The error information encoded as an exception.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ErrorHandler.fatalError\(org.xml.sax.SAXParseException\)](#),
[SAXParseException](#)

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Interface DocumentHandler

All Known Implementing Classes:

[HandlerBase](#), [ParserAdapter](#)

Deprecated. *This interface has been replaced by the SAX2 [ContentHandler](#) interface, which includes Namespace support.*

public abstract interface **DocumentHandler**

Receive notification of general document events.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This was the main event-handling interface for SAX1; in SAX2, it has been replaced by [ContentHandler](#), which provides Namespace support and reporting of skipped entities. This interface is included in SAX2 only to support legacy SAX1 applications.

The order of events in this interface is very important, and mirrors the order of information in the document itself. For example, all of an element's content (character data, processing instructions, and/or subelements) will appear, in order, between the startElement event and the corresponding endElement event.

Application writers who do not want to implement the entire interface can derive a class from HandlerBase, which implements the default functionality; parser writers can instantiate HandlerBase to obtain a default handler. The application can find the location of any document event using the Locator interface supplied by the Parser through the setDocumentLocator method.

Since:

SAX 1.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[Parser.setDocumentHandler\(org.xml.sax.DocumentHandler\)](#), [Locator](#), [HandlerBase](#)

Method Summary

void	<code>characters</code> (char[] ch, int start, int length) Deprecated. Receive notification of character data.
void	<code>endDocument</code> () Deprecated. Receive notification of the end of a document.
void	<code>endElement</code> (java.lang.String name) Deprecated. Receive notification of the end of an element.
void	<code>ignorableWhitespace</code> (char[] ch, int start, int length) Deprecated. Receive notification of ignorable whitespace in element content.
void	<code>processingInstruction</code> (java.lang.String target, java.lang.String data) Deprecated. Receive notification of a processing instruction.
void	<code>setDocumentLocator</code> (<code>Locator</code> locator) Deprecated. Receive an object for locating the origin of SAX document events.
void	<code>startDocument</code> () Deprecated. Receive notification of the beginning of a document.
void	<code>startElement</code> (java.lang.String name, <code>AttributeList</code> atts) Deprecated. Receive notification of the beginning of an element.

Method Detail

setDocumentLocator

```
public void setDocumentLocator(Locator locator)
```

Deprecated.

Receive an object for locating the origin of SAX document events.

SAX parsers are strongly encouraged (though not absolutely required) to supply a locator: if it does so, it must supply the locator to the application by invoking this method before invoking any of the other methods in the DocumentHandler interface.

The locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application will use this information for reporting its own errors (such as character content that does not match an application's business rules). The information returned by the locator is probably not sufficient for use with a search engine.

Note that the locator will return correct information only during the invocation of the events in this

interface. The application should not attempt to use it at any other time.

Parameters:

`locator` - An object that can return the location of any SAX document event.

See Also:

[Locator](#)

startDocument

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

Deprecated.

Receive notification of the beginning of a document.

The SAX parser will invoke this method only once, before any other methods in this interface or in DTDHandler (except for setDocumentLocator).

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

endDocument

```
public void endDocument()  
           throws SAXException
```

Deprecated.

Receive notification of the end of a document.

The SAX parser will invoke this method only once, and it will be the last method invoked during the parse. The parser shall not invoke this method until it has either abandoned parsing (because of an unrecoverable error) or reached the end of input.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

startElement

```
public void startElement(java.lang.String name,  
                        AttributeList atts)  
           throws SAXException
```

Deprecated.

Receive notification of the beginning of an element.

The Parser will invoke this method at the beginning of every element in the XML document; there will be a corresponding `endElement()` event for every `startElement()` event (even when the element is empty). All of the element's content will be reported, in order, before the corresponding `endElement()` event.

If the element name has a namespace prefix, the prefix will still be attached. Note that the attribute list provided will contain only attributes with explicit values (specified or defaulted): #IMPLIED attributes will be omitted.

Parameters:

`name` - The element type name.

`atts` - The attributes attached to the element, if any.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[endElement\(java.lang.String\)](#), [AttributeList](#)

endElement

```
public void endElement(java.lang.String name)
           throws SAXException
```

Deprecated.

Receive notification of the end of an element.

The SAX parser will invoke this method at the end of every element in the XML document; there will be a corresponding `startElement()` event for every `endElement()` event (even when the element is empty).

If the element name has a namespace prefix, the prefix will still be attached to the name.

Parameters:

`name` - The element type name

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

characters

```
public void characters(char[] ch,
```

```
        int start,  
        int length)  
    throws SAXException
```

Deprecated.

Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

The application must not attempt to read from the array outside of the specified range.

Note that some parsers will report whitespace using the `ignoreableWhitespace()` method rather than this one (validating parsers must do so).

Parameters:

`ch` - The characters from the XML document.

`start` - The start position in the array.

`length` - The number of characters to read from the array.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[ignoreableWhitespace\(char\[\], int, int\)](#), [Locator](#)

ignoreableWhitespace

```
public void ignoreableWhitespace(char[] ch,  
                                int start,  
                                int length)  
    throws SAXException
```

Deprecated.

Receive notification of ignoreable whitespace in element content.

Validating Parsers must use this method to report each chunk of ignoreable whitespace (see the W3C XML 1.0 recommendation, section 2.10): non-validating parsers may also use this method if they are capable of parsing and using content models.

SAX parsers may return all contiguous whitespace in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

The application must not attempt to read from the array outside of the specified range.

Parameters:

`ch` - The characters from the XML document.

`start` - The start position in the array.

`length` - The number of characters to read from the array.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

See Also:

[characters\(char\[\], int, int\)](#)

processingInstruction

```
public void processingInstruction(java.lang.String target,
                                   java.lang.String data)
    throws SAXException
```

Deprecated.

Receive notification of a processing instruction.

The Parser will invoke this method once for each processing instruction found: note that processing instructions may occur before or after the main document element.

A SAX parser should never report an XML declaration (XML 1.0, section 2.8) or a text declaration (XML 1.0, section 4.3.1) using this method.

Parameters:

`target` - The processing instruction target.

`data` - The processing instruction data, or null if none was supplied.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Class InputSource

java.lang.Object

|

+--org.xml.sax.InputSource

public class **InputSource**

extends java.lang.Object

A single input source for an XML entity.

This module, both source code and documentation, is in the Public Domain, and comes with NO WARRANTY.

This class allows a SAX application to encapsulate information about an input source in a single object, which may include a public identifier, a system identifier, a byte stream (possibly with a specified encoding), and/or a character stream.

There are two places that the application will deliver this input source to the parser: as the argument to the Parser.parse method, or as the return value of the EntityResolver.resolveEntity method.

The SAX parser will use the InputSource object to determine how to read XML input. If there is a character stream available, the parser will read that stream directly; if not, the parser will use a byte stream, if available; if neither a character stream nor a byte stream is available, the parser will attempt to open a URI connection to the resource identified by the system identifier.

An InputSource object belongs to the application: the SAX parser shall never modify it in any way (it may modify a copy if necessary).

Since:

SAX 1.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[Parser.parse\(org.xml.sax.InputSource\)](#),
[EntityResolver.resolveEntity\(java.lang.String, java.lang.String\)](#),
 InputStream, Reader

Constructor Summary

[InputSource](#)()

Zero-argument default constructor.

[InputSource](#)(java.io.InputStream byteStream)

Create a new input source with a byte stream.

[InputSource](#)(java.io.Reader characterStream)

Create a new input source with a character stream.

[InputSource](#)(java.lang.String systemId)

Create a new input source with a system identifier.

Method Summary

java.io.InputStream [getByteStream](#)()

Get the byte stream for this input source.

java.io.Reader [getCharacterStream](#)()

Get the character stream for this input source.

java.lang.String [getEncoding](#)()

Get the character encoding for a byte stream or URI.

java.lang.String [getPublicId](#)()

Get the public identifier for this input source.

java.lang.String [getSystemId](#)()

Get the system identifier for this input source.

void [setByteStream](#)(java.io.InputStream byteStream)

Set the byte stream for this input source.

void [setCharacterStream](#)(java.io.Reader characterStream)

Set the character stream for this input source.

void [setEncoding](#)(java.lang.String encoding)

Set the character encoding, if known.

void [setPublicId](#)(java.lang.String publicId)

Set the public identifier for this input source.

void [setSystemId](#)(java.lang.String systemId)

Set the system identifier for this input source.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

InputSource

public InputSource()

Zero-argument default constructor.

See Also:

[setPublicId\(java.lang.String\)](#), [setSystemId\(java.lang.String\)](#),
[setByteStream\(java.io.InputStream\)](#),
[setCharacterStream\(java.io.Reader\)](#),
[setEncoding\(java.lang.String\)](#)

InputSource

public InputSource(java.lang.String systemId)

Create a new input source with a system identifier.

Applications may use setPublicId to include a public identifier as well, or setEncoding to specify the character encoding, if known.

If the system identifier is a URL, it must be full resolved.

Parameters:

systemId - The system identifier (URI).

See Also:

[setPublicId\(java.lang.String\)](#), [setSystemId\(java.lang.String\)](#),
[setByteStream\(java.io.InputStream\)](#),
[setEncoding\(java.lang.String\)](#),
[setCharacterStream\(java.io.Reader\)](#)

InputSource

public InputSource(java.io.InputStream byteStream)

Create a new input source with a byte stream.

Application writers may use `setSystemId` to provide a base for resolving relative URIs, `setPublicId` to include a public identifier, and/or `setEncoding` to specify the object's character encoding.

Parameters:

`byteStream` - The raw byte stream containing the document.

See Also:

[setPublicId\(java.lang.String\)](#), [setSystemId\(java.lang.String\)](#),
[setEncoding\(java.lang.String\)](#),
[setByteStream\(java.io.InputStream\)](#),
[setCharacterStream\(java.io.Reader\)](#)

InputSource

```
public InputSource(java.io.Reader characterStream)
```

Create a new input source with a character stream.

Application writers may use `setSystemId()` to provide a base for resolving relative URIs, and `setPublicId` to include a public identifier.

The character stream shall not include a byte order mark.

See Also:

[setPublicId\(java.lang.String\)](#), [setSystemId\(java.lang.String\)](#),
[setByteStream\(java.io.InputStream\)](#),
[setCharacterStream\(java.io.Reader\)](#)

Method Detail

setPublicId

```
public void setPublicId(java.lang.String publicId)
```

Set the public identifier for this input source.

The public identifier is always optional: if the application writer includes one, it will be provided as part of the location information.

Parameters:

`publicId` - The public identifier as a string.

See Also:

[getPublicId\(\)](#), [Locator.getPublicId\(\)](#),
[SAXParseException.getPublicId\(\)](#)

getPublicId

```
public java.lang.String getPublicId()
```

Get the public identifier for this input source.

Returns:

The public identifier, or null if none was supplied.

See Also:

[setPublicId\(java.lang.String\)](#)

setSystemId

```
public void setSystemId(java.lang.String systemId)
```

Set the system identifier for this input source.

The system identifier is optional if there is a byte stream or a character stream, but it is still useful to provide one, since the application can use it to resolve relative URIs and can include it in error messages and warnings (the parser will attempt to open a connection to the URI only if there is no byte stream or character stream specified).

If the application knows the character encoding of the object pointed to by the system identifier, it can register the encoding using the `setEncoding` method.

If the system ID is a URL, it must be fully resolved.

Parameters:

`systemId` - The system identifier as a string.

See Also:

[setEncoding\(java.lang.String\)](#), [getSystemId\(\)](#),
[Locator.getSystemId\(\)](#), [SAXParseException.getSystemId\(\)](#)

getSystemId

```
public java.lang.String getSystemId()
```

Get the system identifier for this input source.

The `getEncoding` method will return the character encoding of the object pointed to, or null if unknown.

If the system ID is a URL, it will be fully resolved.

Returns:

The system identifier.

See Also:

[setSystemId\(java.lang.String\)](#), [getEncoding\(\)](#)

setByteStream

```
public void setByteStream(java.io.InputStream byteStream)
```

Set the byte stream for this input source.

The SAX parser will ignore this if there is also a character stream specified, but it will use a byte stream in preference to opening a URI connection itself.

If the application knows the character encoding of the byte stream, it should set it with the `setEncoding` method.

Parameters:

`byteStream` - A byte stream containing an XML document or other entity.

See Also:

[setEncoding\(java.lang.String\)](#), [getByteStream\(\)](#), [getEncoding\(\)](#), `InputStream`

getByteStream

```
public java.io.InputStream getByteStream()
```

Get the byte stream for this input source.

The `getEncoding` method will return the character encoding for this byte stream, or null if unknown.

Returns:

The byte stream, or null if none was supplied.

See Also:

[getEncoding\(\)](#), [setByteStream\(java.io.InputStream\)](#)

setEncoding

```
public void setEncoding(java.lang.String encoding)
```

Set the character encoding, if known.

The encoding must be a string acceptable for an XML encoding declaration (see section 4.3.3 of the XML 1.0 recommendation).

This method has no effect when the application provides a character stream.

Parameters:

encoding - A string describing the character encoding.

See Also:

[setSystemId\(java.lang.String\)](#),
[setByteStream\(java.io.InputStream\)](#), [getEncoding\(\)](#)

getEncoding

```
public java.lang.String getEncoding()
```

Get the character encoding for a byte stream or URI.

Returns:

The encoding, or null if none was supplied.

See Also:

[setByteStream\(java.io.InputStream\)](#), [getSystemId\(\)](#),
[getByteStream\(\)](#)

setCharacterStream

```
public void setCharacterStream(java.io.Reader characterStream)
```

Set the character stream for this input source.

If there is a character stream specified, the SAX parser will ignore any byte stream and will not attempt to open a URI connection to the system identifier.

Parameters:

characterStream - The character stream containing the XML document or other entity.

See Also:

[getCharacterStream\(\)](#), [Reader](#)

getCharacterStream

```
public java.io.Reader getCharacterStream()
```

Get the character stream for this input source.

Returns:

The character stream, or null if none was supplied.

See Also:

[setCharacterStream\(java.io.Reader\)](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax.helpers

Class LocatorImpl

```
java.lang.Object
|
+--org.xml.sax.helpers.LocatorImpl
```

public class **LocatorImpl**

extends java.lang.Object

implements [Locator](#)

Provide an optional convenience implementation of Locator.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This class is available mainly for application writers, who can use it to make a persistent snapshot of a locator at any point during a document parse:

```
Locator locator;
Locator startloc;

public void setLocator (Locator locator)
{
    // note the locator
    this.locator = locator;
}

public void startDocument ()
{
    // save the location of the start of the document
    // for future use.
    Locator startloc = new LocatorImpl(locator);
}
```

Normally, parser writers will not use this class, since it is more efficient to provide location information only when requested, rather than constantly updating a Locator object.

Since:

SAX 1.0

Version:

2.0

Author:David Megginson, sax@megginson.com**See Also:**[Locator](#)

Constructor Summary

[LocatorImpl](#)()

Zero-argument constructor.

[LocatorImpl](#)([Locator](#) locator)

Copy constructor.

Method Summary

int	getColumnNumber ()	Return the saved column number (1-based).
int	getLineNumber ()	Return the saved line number (1-based).
java.lang.String	getPublicId ()	Return the saved public identifier.
java.lang.String	getSystemId ()	Return the saved system identifier.
void	setColumnNumber (int columnNumber)	Set the column number for this locator (1-based).
void	setLineNumber (int lineNumber)	Set the line number for this locator (1-based).
void	setPublicId (java.lang.String publicId)	Set the public identifier for this locator.
void	setSystemId (java.lang.String systemId)	Set the system identifier for this locator.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

LocatorImpl

```
public LocatorImpl()
```

Zero-argument constructor.

This will not normally be useful, since the main purpose of this class is to make a snapshot of an existing Locator.

LocatorImpl

```
public LocatorImpl(Locator locator)
```

Copy constructor.

Create a persistent copy of the current state of a locator. When the original locator changes, this copy will still keep the original values (and it can be used outside the scope of `DocumentHandler` methods).

Parameters:

`locator` - The locator to copy.

Method Detail

getPublicId

```
public java.lang.String getPublicId()
```

Return the saved public identifier.

Specified by:

[getPublicId](#) in interface [Locator](#)

Returns:

The public identifier as a string, or null if none is available.

See Also:

[Locator.getPublicId\(\)](#), [setPublicId\(java.lang.String\)](#)

getSystemId

```
public java.lang.String getSystemId()
```

Return the saved system identifier.

Specified by:

[getSystemId](#) in interface [Locator](#)

Returns:

The system identifier as a string, or null if none is available.

See Also:

[Locator.getSystemId\(\)](#), [setSystemId\(java.lang.String\)](#)

getLineNumber

```
public int getLineNumber()
```

Return the saved line number (1-based).

Specified by:

[getLineNumber](#) in interface [Locator](#)

Returns:

The line number as an integer, or -1 if none is available.

See Also:

[Locator.getLineNumber\(\)](#), [setLineNumber\(int\)](#)

getColumnNumber

```
public int getColumnNumber()
```

Return the saved column number (1-based).

Specified by:

[getColumnNumber](#) in interface [Locator](#)

Returns:

The column number as an integer, or -1 if none is available.

See Also:

[Locator.getColumnNumber\(\)](#), [setColumnNumber\(int\)](#)

setPublicId

```
public void setPublicId(java.lang.String publicId)
```

Set the public identifier for this locator.

Parameters:

publicId - The new public identifier, or null if none is available.

See Also:

[getPublicId\(\)](#)

setSystemId

```
public void setSystemId(java.lang.String systemId)
```

Set the system identifier for this locator.

Parameters:

systemId - The new system identifier, or null if none is available.

See Also:

[getSystemId\(\)](#)

setLineNumber

```
public void setLineNumber(int lineNumber)
```

Set the line number for this locator (1-based).

Parameters:

lineNumber - The line number, or -1 if none is available.

See Also:

[getLineNumber\(\)](#)

setColumnNumber

```
public void setColumnNumber(int columnNumber)
```

Set the column number for this locator (1-based).

Parameters:

columnNumber - The column number, or -1 if none is available.

See Also:

[getColumnNumber\(\)](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: INNER | FIELD | CONSTR | [METHOD](#)DETAIL: FIELD | CONSTR | [METHOD](#)**org.xml.sax**

Interface Locator

All Known Implementing Classes:

[LocatorImpl](#)

public abstract interface **Locator**

Interface for associating a SAX event with a document location.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

If a SAX parser provides location information to the SAX application, it does so by implementing this interface and then passing an instance to the application using the content handler's [setDocumentLocator](#) method. The application can use the object to obtain the location of any other content handler event in the XML source document.

Note that the results returned by the object will be valid only during the scope of each content handler method: the application will receive unpredictable results if it attempts to use the locator at any other time.

SAX parsers are not required to supply a locator, but they are very strongly encouraged to do so. If the parser supplies a locator, it must do so before reporting any other document events. If no locator has been set by the time the application receives the [startDocument](#) event, the application should assume that a locator is not available.

Since:

SAX 1.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[ContentHandler.setDocumentLocator\(org.xml.sax.Locator\)](#)

Method Summary

int	getColumnNumber ()
-----	-------------------------------------

Return the column number where the current document event ends.

int	<code>getLineNumber()</code> Return the line number where the current document event ends.
java.lang.String	<code>getPublicId()</code> Return the public identifier for the current document event.
java.lang.String	<code>getSystemId()</code> Return the system identifier for the current document event.

Method Detail

getPublicId

```
public java.lang.String getPublicId()
```

Return the public identifier for the current document event.

The return value is the public identifier of the document entity or of the external parsed entity in which the markup triggering the event appears.

Returns:

A string containing the public identifier, or null if none is available.

See Also:

[`getSystemId\(\)`](#)

getSystemId

```
public java.lang.String getSystemId()
```

Return the system identifier for the current document event.

The return value is the system identifier of the document entity or of the external parsed entity in which the markup triggering the event appears.

If the system identifier is a URL, the parser must resolve it fully before passing it to the application.

Returns:

A string containing the system identifier, or null if none is available.

See Also:

[`getPublicId\(\)`](#)

getLineNumber

```
public int getLineNumber()
```

Return the line number where the current document event ends.

Warning: The return value from the method is intended only as an approximation for the sake of error reporting; it is not intended to provide sufficient information to edit the character content of the original XML document.

The return value is an approximation of the line number in the document entity or external parsed entity where the markup triggering the event appears.

If possible, the SAX driver should provide the line position of the first character after the text associated with the document event. The first line in the document is line 1.

Returns:

The line number, or -1 if none is available.

See Also:

[getColumnNumber\(\)](#)

getColumnNumber

```
public int getColumnNumber()
```

Return the column number where the current document event ends.

Warning: The return value from the method is intended only as an approximation for the sake of error reporting; it is not intended to provide sufficient information to edit the character content of the original XML document.

The return value is an approximation of the column number in the document entity or external parsed entity where the markup triggering the event appears.

If possible, the SAX driver should provide the line position of the first character after the text associated with the document event.

If possible, the SAX driver should provide the line position of the first character after the text associated with the document event. The first column in each line is column 1.

Returns:

The column number, or -1 if none is available.

See Also:

[getLineNumber\(\)](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)
org.xml.sax.helpers

Class NamespaceSupport

java.lang.Object

```

|
+--org.xml.sax.helpers.NamespaceSupport

```

public class NamespaceSupport

extends java.lang.Object

Encapsulate Namespace logic for use by SAX drivers.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This class encapsulates the logic of Namespace processing: it tracks the declarations currently in force for each context and automatically processes qualified XML 1.0 names into their Namespace parts; it can also be used in reverse for generating XML 1.0 from Namespaces.

Namespace support objects are reusable, but the reset method must be invoked between each session.

Here is a simple session:

```

String parts[] = new String[3];
NamespaceSupport support = new NamespaceSupport();

support.pushContext();
support.declarePrefix("", "http://www.w3.org/1999/xhtml");
support.declarePrefix("dc", "http://www.purl.org/dc#");

String parts[] = support.processName("p", parts, false);
System.out.println("Namespace URI: " + parts[0]);
System.out.println("Local name: " + parts[1]);
System.out.println("Raw name: " + parts[2]);

String parts[] = support.processName("dc:title", parts, false);
System.out.println("Namespace URI: " + parts[0]);
System.out.println("Local name: " + parts[1]);
System.out.println("Raw name: " + parts[2]);

support.popContext();

```

Note that this class is optimized for the use case where most elements do not contain Namespace declarations: if the same prefix/URI mapping is repeated for each context (for example), this class will be somewhat less efficient.

Since:

SAX 2.0

Version:

2.0

Author:David Megginson, sax@megginson.com

Field Summary

static java.lang.String	<u>XMLNS</u>
-------------------------	------------------------------

	The XML Namespace as a constant.
--	----------------------------------

Constructor Summary

<u>NamespaceSupport</u> ()

Create a new Namespace support object.
--

Method Summary

boolean	<u>declarePrefix</u> (java.lang.String prefix, java.lang.String uri) Declare a Namespace prefix.
---------	---

java.util.Enumeration	<u>getDeclaredPrefixes</u> () Return an enumeration of all prefixes declared in this context.
-----------------------	---

java.lang.String	<u>getPrefix</u> (java.lang.String uri) Return one of the prefixes mapped to a Namespace URI.
------------------	---

java.util.Enumeration	<u>getPrefixes</u> () Return an enumeration of all prefixes currently declared.
-----------------------	---

java.util.Enumeration	<u>getPrefixes</u> (java.lang.String uri) Return an enumeration of all prefixes currently declared for a URI.
-----------------------	---

java.lang.String	<u>getURI</u> (java.lang.String prefix) Look up a prefix and get the currently-mapped Namespace URI.
------------------	--

void	<u>popContext</u> () Revert to the previous Namespace context.
------	--

java.lang.String[]	processName (java.lang.String qName, java.lang.String[] parts, boolean isAttribute) Process a raw XML 1.0 name.
void	pushContext () Start a new Namespace context.
void	reset () Reset this Namespace support object for reuse.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

XMLNS

```
public static final java.lang.String XMLNS
```

The XML Namespace as a constant.

This is the Namespace URI that is automatically mapped to the "xml" prefix.

Constructor Detail

NamespaceSupport

```
public NamespaceSupport( )
```

Create a new Namespace support object.

Method Detail

reset

```
public void reset( )
```

Reset this Namespace support object for reuse.

It is necessary to invoke this method before reusing the Namespace support object for a new session.

pushContext

```
public void pushContext()
```

Start a new Namespace context.

Normally, you should push a new context at the beginning of each XML element: the new context will automatically inherit the declarations of its parent context, but it will also keep track of which declarations were made within this context.

The Namespace support object always starts with a base context already in force: in this context, only the "xml" prefix is declared.

See Also:

[`popContext\(\)`](#)

popContext

```
public void popContext()
```

Revert to the previous Namespace context.

Normally, you should pop the context at the end of each XML element. After popping the context, all Namespace prefix mappings that were previously in force are restored.

You must not attempt to declare additional Namespace prefixes after popping a context, unless you push another context first.

See Also:

[`pushContext\(\)`](#)

declarePrefix

```
public boolean declarePrefix(java.lang.String prefix,  
                               java.lang.String uri)
```

Declare a Namespace prefix.

This method declares a prefix in the current Namespace context; the prefix will remain in force until this context is popped, unless it is shadowed in a descendant context.

To declare a default Namespace, use the empty string. The prefix must not be "xml" or "xmlns".

Note that you must *not* declare a prefix after you've pushed and popped another Namespace.

Note that there is an asymmetry in this library: while [getPrefix](#) will not return the default "" prefix, even if you have declared one; to check for a default prefix, you have to look it up explicitly using [getURI](#). This asymmetry exists to make it easier to look up prefixes for attribute names, where the default prefix is not allowed.

Parameters:

`prefix` - The prefix to declare, or null for the empty string.

`uri` - The Namespace URI to associate with the prefix.

Returns:

true if the prefix was legal, false otherwise

See Also:

[processName\(java.lang.String, java.lang.String\[\], boolean\)](#),
[getURI\(java.lang.String\)](#), [getPrefix\(java.lang.String\)](#)

processName

```
public java.lang.String[] processName(java.lang.String qName,
                                       java.lang.String[] parts,
                                       boolean isAttribute)
```

Process a raw XML 1.0 name.

This method processes a raw XML 1.0 name in the current context by removing the prefix and looking it up among the prefixes currently declared. The return value will be the array supplied by the caller, filled in as follows:

`parts[0]`

The Namespace URI, or an empty string if none is in use.

`parts[1]`

The local name (without prefix).

`parts[2]`

The original raw name.

All of the strings in the array will be internalized. If the raw name has a prefix that has not been declared, then the return value will be null.

Note that attribute names are processed differently than element names: an unprefix element name will received the default Namespace (if any), while an unprefix element name will not.

Parameters:

`qName` - The raw XML 1.0 name to be processed.

`parts` - An array supplied by the caller, capable of holding at least three members.

`isAttribute` - A flag indicating whether this is an attribute name (true) or an element name (false).

Returns:

The supplied array holding three internalized strings representing the Namespace URI (or empty string), the local name, and the raw XML 1.0 name; or null if there is an undeclared prefix.

See Also:

[declarePrefix\(java.lang.String, java.lang.String\)](#),
[String.intern\(\)](#)

getURI

```
public java.lang.String getURI(java.lang.String prefix)
```

Look up a prefix and get the currently-mapped Namespace URI.

This method looks up the prefix in the current context. Use the empty string ("") for the default Namespace.

Parameters:

`prefix` - The prefix to look up.

Returns:

The associated Namespace URI, or null if the prefix is undeclared in this context.

See Also:

[getPrefix\(java.lang.String\)](#), [getPrefixes\(\)](#)

getPrefixes

```
public java.util Enumeration getPrefixes()
```

Return an enumeration of all prefixes currently declared.

Note: if there is a default prefix, it will not be returned in this enumeration; check for the default prefix using the [getURI](#) with an argument of "".

Returns:

An enumeration of all prefixes declared in the current context except for the empty (default) prefix.

See Also:

[getDeclaredPrefixes\(\)](#), [getURI\(java.lang.String\)](#)

getPrefix

```
public java.lang.String getPrefix(java.lang.String uri)
```

Return one of the prefixes mapped to a Namespace URI.

If more than one prefix is currently mapped to the same URI, this method will make an arbitrary selection; if you want all of the prefixes, use the [getPrefixes\(\)](#) method instead.

Note: this will never return the empty (default) prefix; to check for a default prefix, use the [getURI](#) method with an argument of "".

Parameters:

`uri` - The Namespace URI.

`isAttribute` - true if this prefix is for an attribute (and the default Namespace is not allowed).

Returns:

One of the prefixes currently mapped to the URI supplied, or null if none is mapped or if the URI is assigned to the default Namespace.

See Also:

[getPrefixes\(java.lang.String\)](#), [getURI\(java.lang.String\)](#)

getPrefixes

```
public java.util Enumeration getPrefixes(java.lang.String uri)
```

Return an enumeration of all prefixes currently declared for a URI.

This method returns prefixes mapped to a specific Namespace URI. The xml: prefix will be included. If you want only one prefix that's mapped to the Namespace URI, and you don't care which one you get, use the [getPrefix](#) method instead.

Note: the empty (default) prefix is *never* included in this enumeration; to check for the presence of a default Namespace, use the [getURI](#) method with an argument of "".

Parameters:

`uri` - The Namespace URI.

Returns:

An enumeration of all prefixes declared in the current context.

See Also:

[getPrefix\(java.lang.String\)](#), [getDeclaredPrefixes\(\)](#),
[getURI\(java.lang.String\)](#)

getDeclaredPrefixes

```
public java.util Enumeration getDeclaredPrefixes()
```

Return an enumeration of all prefixes declared in this context.

The empty (default) prefix will be included in this enumeration; note that this behaviour differs from that of [getPrefix\(java.lang.String\)](#) and [getPrefixes\(\)](#).

Returns:

An enumeration of all prefixes declared in this context.

See Also:

[getPrefixes\(\)](#), [getURI\(java.lang.String\)](#)

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)

 SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

 DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform

Class OutputKeys

java.lang.Object

|

+-- **javax.xml.transform.OutputKeys**public class **OutputKeys**

extends java.lang.Object

Provides string constants that can be used to set output properties for a Transformer, or to retrieve output properties from a Transformer or Templates object.

A properties in this class are read-only.

See Also:

[section 16 of the XSL Transformations \(XSLT\) W3C Recommendation](#)

Field Summary

static java.lang.String	CDATA_SECTION_ELEMENTS cdata-section-elements = expanded names.
static java.lang.String	DOCTYPE_PUBLIC doctype-public = string.
static java.lang.String	DOCTYPE_SYSTEM doctype-system = string.
static java.lang.String	ENCODING encoding = string.
static java.lang.String	INDENT indent = "yes" "no".
static java.lang.String	MEDIA_TYPE media-type = string.
static java.lang.String	METHOD method = "xml" "html" "text" expanded name.
static java.lang.String	OMIT_XML_DECLARATION omit-xml-declaration = "yes" "no".

static java.lang.String	STANDALONE standalone = "yes" "no".
static java.lang.String	VERSION version = nmtoken.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

METHOD

```
public static final java.lang.String METHOD
    method = "xml" | "html" | "text" | expanded name.
```

The method attribute identifies the overall method that should be used for outputting the result tree. Other non-namespaced values may be used, such as "xhtml", but, if accepted, the handling of such values is implementation defined. If any of the method values are not accepted and are not namespace qualified, then [Transformer.setOutputProperty\(java.lang.String, java.lang.String\)](#) or [Transformer.setOutputProperties\(java.util.Properties\)](#) will throw a `IllegalArgumentException`.

See Also:

[section 16 of the XSL Transformations \(XSLT\) W3C Recommendation](#)

VERSION

```
public static final java.lang.String VERSION
    version = nmtoken.
```

version specifies the version of the output method.

When the output method is "xml", the version value specifies the version of XML to be used for outputting the result tree. The default value for the xml output method is 1.0. When the output method is "html", the version value indicates the version of the HTML. The default value for the xml output method is 4.0, which specifies that the result should be output as HTML conforming to the HTML 4.0 Recommendation [HTML]. If the output method is "text", the version property is

ignored.

See Also:

[section 16 of the XSL Transformations \(XSLT\) W3C Recommendation](#)

ENCODING

```
public static final java.lang.String ENCODING  
    encoding = string.
```

`encoding` specifies the preferred character encoding that the Transformer should use to encode sequences of characters as sequences of bytes. The value of the attribute should be treated case-insensitively. The value must only contain characters in the range #x21 to #x7E (i.e., printable ASCII characters). The value should either be a `charset` registered with the Internet Assigned Numbers Authority [\[IANA\]](#), [\[RFC2278\]](#) or start with X-.

See Also:

[section 16 of the XSL Transformations \(XSLT\) W3C Recommendation](#)

OMIT_XML_DECLARATION

```
public static final java.lang.String OMIT_XML_DECLARATION  
    omit-xml-declaration = "yes" | "no".
```

`omit-xml-declaration` specifies whether the XSLT processor should output an XML declaration; the value must be `yes` or `no`.

See Also:

[section 16 of the XSL Transformations \(XSLT\) W3C Recommendation](#)

STANDALONE

```
public static final java.lang.String STANDALONE  
    standalone = "yes" | "no".
```

`standalone` specifies whether the Transformer should output a standalone document declaration; the value must be `yes` or `no`.

See Also:

[section 16 of the XSL Transformations \(XSLT\) W3C Recommendation](#)

DOCTYPE_PUBLIC

```
public static final java.lang.String DOCTYPE_PUBLIC
doctype-public = string.
```

See the documentation for the [DOCTYPE_SYSTEM](#) property for a description of what the value of the key should be.

See Also:

[section 16 of the XSL Transformations \(XSLT\) W3C Recommendation](#)

DOCTYPE_SYSTEM

```
public static final java.lang.String DOCTYPE_SYSTEM
doctype-system = string.
```

`doctype-public` specifies the public identifier to be used in the document type declaration.

If the `doctype-system` property is specified, the xml output method should output a document type declaration immediately before the first element. The name following `<!DOCTYPE` should be the name of the first element. If `doctype-public` property is also specified, then the xml output method should output PUBLIC followed by the public identifier and then the system identifier; otherwise, it should output SYSTEM followed by the system identifier. The internal subset should be empty. The `doctype-public` attribute should be ignored unless the `doctype-system` attribute is specified.

If the `doctype-public` or `doctype-system` attributes are specified, then the html output method should output a document type declaration immediately before the first element. The name following `<!DOCTYPE` should be HTML or html. If the `doctype-public` attribute is specified, then the output method should output PUBLIC followed by the specified public identifier; if the `doctype-system` attribute is also specified, it should also output the specified system identifier following the public identifier. If the `doctype-system` attribute is specified but the `doctype-public` attribute is not specified, then the output method should output SYSTEM followed by the specified system identifier.

`doctype-system` specifies the system identifier to be used in the document type declaration.

See Also:

[section 16 of the XSL Transformations \(XSLT\) W3C Recommendation](#)

CDATA_SECTION_ELEMENTS

```
public static final java.lang.String CDATA_SECTION_ELEMENTS
```

`cdata-section-elements` = expanded names.

`cdata-section-elements` specifies a whitespace delimited list of the names of elements whose text node children should be output using CDATA sections.

See Also:

[section 16 of the XSL Transformations \(XSLT\) W3C Recommendation.](#)

INDENT

```
public static final java.lang.String INDENT
    indent = "yes" | "no".
```

`indent` specifies whether the Transformer may add additional whitespace when outputting the result tree; the value must be `yes` or `no`.

See Also:

[section 16 of the XSL Transformations \(XSLT\) W3C Recommendation](#)

MEDIA_TYPE

```
public static final java.lang.String MEDIA_TYPE
    media-type = string.
```

`media-type` specifies the media type (MIME content type) of the data that results from outputting the result tree. The `charset` parameter should not be specified explicitly; instead, when the top-level media type is `text`, a `charset` parameter should be added according to the character encoding actually used by the output method.

See Also:

[section 16 of the XSL Transformations \(XSLT\) W3C Recommendation](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax.helpers

Class ParserAdapter

java.lang.Object

```

|
+--org.xml.sax.helpers.ParserAdapter

```

public class **ParserAdapter**

extends java.lang.Object

implements [XMLReader](#), [DocumentHandler](#)

Adapt a SAX1 Parser as a SAX2 XMLReader.

This module, both source code and documentation, is in the Public Domain, and comes with NO WARRANTY.

This class wraps a SAX1 [Parser](#) and makes it act as a SAX2 [XMLReader](#), with feature, property, and Namespace support. Note that it is not possible to report [skippedEntity](#) events, since SAX1 does not make that information available.

This adapter does not test for duplicate Namespace-qualified attribute names.

Since:

SAX 2.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[XMLReaderAdapter](#), [XMLReader](#), [Parser](#)

Constructor Summary

[ParserAdapter](#) ()

Construct a new parser adapter.

[ParserAdapter](#) ([Parser](#) parser)

Construct a new parser adapter.

Method Summary

void	characters (char[] ch, int start, int length) Adapt a SAX1 characters event.
void	endDocument () Adapt a SAX1 end document event.
void	endElement (java.lang.String qName) Adapt a SAX1 end element event.
ContentHandler	getContentHandler () Return the current content handler.
DTDHandler	getDTDHandler () Return the current DTD handler.
EntityResolver	getEntityResolver () Return the current entity resolver.
ErrorHandler	getErrorHandler () Return the current error handler.
boolean	getFeature (java.lang.String name) Check a parser feature.
java.lang.Object	getProperty (java.lang.String name) Get a parser property.
void	ignorableWhitespace (char[] ch, int start, int length) Adapt a SAX1 ignorable whitespace event.
void	parse (InputSource input) Parse an XML document.
void	parse (java.lang.String systemId) Parse an XML document.
void	processingInstruction (java.lang.String target, java.lang.String data) Adapt a SAX1 processing instruction event.
void	setContentHandler (ContentHandler handler) Set the content handler.
void	setDocumentLocator (Locator locator) Adapt a SAX1 document locator event.
void	setDTDHandler (DTDHandler handler) Set the DTD handler.

void	setEntityResolver (EntityResolver resolver) Set the entity resolver.
void	setErrorHandler (ErrorHandler handler) Set the error handler.
void	setFeature (java.lang.String name, boolean state) Set a feature for the parser.
void	setProperty (java.lang.String name, java.lang.Object value) Set a parser property.
void	startDocument () Adapt a SAX1 start document event.
void	startElement (java.lang.String qName, AttributeList qAtts) Adapt a SAX1 startElement event.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ParserAdapter

```
public ParserAdapter()  
    throws SAXException
```

Construct a new parser adapter.

Use the "org.xml.sax.parser" property to locate the embedded SAX1 driver.

Throws:

[SAXException](#) - If the embedded driver cannot be instantiated or if the org.xml.sax.parser property is not specified.

ParserAdapter

```
public ParserAdapter(Parser parser)
```

Construct a new parser adapter.

Note that the embedded parser cannot be changed once the adapter is created; to embed a different parser, allocate a new ParserAdapter.

Parameters:

`parser` - The SAX1 parser to embed.

Throws:

`java.lang.NullPointerException` - If the parser parameter is null.

Method Detail

setFeature

```
public void setFeature(java.lang.String name,  
                       boolean state)  
    throws SAXNotRecognizedException,  
           SAXNotSupportedException
```

Set a feature for the parser.

The only features supported are namespaces and namespace-prefixes.

Specified by:

[setFeature](#) in interface [XMLReader](#)

Parameters:

`name` - The feature name, as a complete URI.

`state` - The requested feature state.

Throws:

[SAXNotRecognizedException](#) - If the feature name is not known.

[SAXNotSupportedException](#) - If the feature state is not supported.

See Also:

[XMLReader.setFeature\(java.lang.String, boolean\)](#)

getFeature

```
public boolean getFeature(java.lang.String name)  
    throws SAXNotRecognizedException,  
           SAXNotSupportedException
```

Check a parser feature.

The only features supported are namespaces and namespace-prefixes.

Specified by:

[getFeature](#) in interface [XMLReader](#)

Parameters:

name - The feature name, as a complete URI.

Returns:

The current feature state.

Throws:

[SAXNotRecognizedException](#) - If the feature name is not known.

[SAXNotSupportedException](#) - If querying the feature state is not supported.

See Also:

[XMLReader.setFeature\(java.lang.String, boolean\)](#)

setProperty

```
public void setProperty(java.lang.String name,  
                        java.lang.Object value)  
    throws SAXNotRecognizedException,  
           SAXNotSupportedException
```

Set a parser property.

No special properties are currently supported.

Specified by:

[setProperty](#) in interface [XMLReader](#)

Parameters:

name - The property name.

value - The property value.

Throws:

[SAXNotRecognizedException](#) - If the feature name is not known.

[SAXNotSupportedException](#) - If the feature state is not supported.

See Also:

[XMLReader.setProperty\(java.lang.String, java.lang.Object\)](#)

getProperty

```
public java.lang.Object getProperty(java.lang.String name)  
                                throws SAXNotRecognizedException,  
                                       SAXNotSupportedException
```

Get a parser property.

No special properties are currently supported.

Specified by:

[getProperty](#) in interface [XMLReader](#)

Parameters:

name - The property name.

Returns:

The property value.

Throws:

[SAXNotRecognizedException](#) - If the feature name is not known.

[SAXNotSupportedException](#) - If the feature state is not supported.

See Also:

[XMLReader.getProperty\(java.lang.String\)](#)

setEntityResolver

```
public void setEntityResolver(EntityResolver resolver)
```

Set the entity resolver.

Specified by:

[setEntityResolver](#) in interface [XMLReader](#)

Parameters:

resolver - The new entity resolver.

Throws:

java.lang.NullPointerException - If the entity resolver parameter is null.

See Also:

[XMLReader.setEntityResolver\(org.xml.sax.EntityResolver\)](#)

getEntityResolver

```
public EntityResolver getEntityResolver()
```

Return the current entity resolver.

Specified by:

[getEntityResolver](#) in interface [XMLReader](#)

Returns:

The current entity resolver, or null if none was supplied.

See Also:

[XMLReader.getEntityResolver\(\)](#)

setDTDHandler

```
public void setDTDHandler(DTDHandler handler)
```

Set the DTD handler.

Specified by:

[setDTDHandler](#) in interface [XMLReader](#)

Parameters:

resolver - The new DTD handler.

Throws:

java.lang.NullPointerException - If the DTD handler parameter is null.

See Also:

[XMLReader.setEntityResolver\(org.xml.sax.EntityResolver\)](#)

getDTDHandler

```
public DTDHandler getDTDHandler()
```

Return the current DTD handler.

Specified by:

[getDTDHandler](#) in interface [XMLReader](#)

Returns:

The current DTD handler, or null if none was supplied.

See Also:

[XMLReader.getEntityResolver\(\)](#)

setContentHandler

```
public void setContentHandler(ContentHandler handler)
```

Set the content handler.

Specified by:

[setContentHandler](#) in interface [XMLReader](#)

Parameters:

resolver - The new content handler.

Throws:

java.lang.NullPointerException - If the content handler parameter is null.

See Also:

[XMLReader.setEntityResolver\(org.xml.sax.EntityResolver\)](#)

getContentHandler

```
public ContentHandler getContentHandler()
```

Return the current content handler.

Specified by:

[getContentHandler](#) in interface [XMLReader](#)

Returns:

The current content handler, or null if none was supplied.

See Also:

[XMLReader.getEntityResolver\(\)](#)

setErrorHandler

```
public void setErrorHandler(ErrorHandler handler)
```

Set the error handler.

Specified by:

[setErrorHandler](#) in interface [XMLReader](#)

Parameters:

resolver - The new error handler.

Throws:

`java.lang.NullPointerException` - If the error handler parameter is null.

See Also:

[XMLReader.setEntityResolver\(org.xml.sax.EntityResolver\)](#)

getErrorHandler

```
public ErrorHandler getErrorHandler()
```

Return the current error handler.

Specified by:

[getErrorHandler](#) in interface [XMLReader](#)

Returns:

The current error handler, or null if none was supplied.

See Also:

[XMLReader.getEntityResolver\(\)](#)

parse

```
public void parse(java.lang.String systemId)  
    throws java.io.IOException,  
           SAXException
```

Parse an XML document.

Specified by:

[parse](#) in interface [XMLReader](#)

Parameters:

`systemId` - The absolute URL of the document.

Throws:

`java.io.IOException` - If there is a problem reading the raw content of the document.

[SAXException](#) - If there is a problem processing the document.

See Also:

[parse\(org.xml.sax.InputSource\)](#), [Parser.parse\(java.lang.String\)](#)

parse

```
public void parse(InputSource input)
    throws java.io.IOException,
           SAXException
```

Parse an XML document.

Specified by:

[parse](#) in interface [XMLReader](#)

Parameters:

input - An input source for the document.

Throws:

java.io.IOException - If there is a problem reading the raw content of the document.

[SAXException](#) - If there is a problem processing the document.

See Also:

[parse\(java.lang.String\)](#), [Parser.parse\(org.xml.sax.InputSource\)](#)

setDocumentLocator

```
public void setDocumentLocator(Locator locator)
```

Adapt a SAX1 document locator event.

Specified by:

[setDocumentLocator](#) in interface [DocumentHandler](#)

Parameters:

locator - A document locator.

See Also:

[ContentHandler.setDocumentLocator\(org.xml.sax.Locator\)](#)

startDocument

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

Adapt a SAX1 start document event.

Specified by:

[startDocument](#) in interface [DocumentHandler](#)

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[DocumentHandler.startDocument\(\)](#)

endDocument

```
public void endDocument()  
           throws SAXException
```

Adapt a SAX1 end document event.

Specified by:

[endDocument](#) in interface [DocumentHandler](#)

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[DocumentHandler.endDocument\(\)](#)

startElement

```
public void startElement(java.lang.String qName,  
                        AttributeList qAtts)  
           throws SAXException
```

Adapt a SAX1 startElement event.

If necessary, perform Namespace processing.

Specified by:

[startElement](#) in interface [DocumentHandler](#)

Parameters:

qName - The qualified (prefixed) name.

qAtts - The XML 1.0 attribute list (with qnames).

endElement

```
public void endElement(java.lang.String qName)  
           throws SAXException
```

Adapt a SAX1 end element event.

Specified by:

[endElement](#) in interface [DocumentHandler](#)

Parameters:

qName - The qualified (prefixed) name.

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[DocumentHandler.endElement\(java.lang.String\)](#)

characters

```
public void characters(char[] ch,  
                      int start,  
                      int length)  
    throws SAXException
```

Adapt a SAX1 characters event.

Specified by:

[characters](#) in interface [DocumentHandler](#)

Parameters:

ch - An array of characters.

start - The starting position in the array.

length - The number of characters to use.

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[DocumentHandler.characters\(char\[\], int, int\)](#)

ignorableWhitespace

```
public void ignorableWhitespace(char[] ch,  
                                int start,  
                                int length)  
    throws SAXException
```

Adapt a SAX1 ignorable whitespace event.

Specified by:

[ignorableWhitespace](#) in interface [DocumentHandler](#)

Parameters:

`ch` - An array of characters.

`start` - The starting position in the array.

`length` - The number of characters to use.

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[DocumentHandler.ignorableWhitespace\(char\[\], int, int\)](#)

processingInstruction

```
public void processingInstruction(java.lang.String target,
                                   java.lang.String data)
                                   throws SAXException
```

Adapt a SAX1 processing instruction event.

Specified by:

[processingInstruction](#) in interface [DocumentHandler](#)

Parameters:

`target` - The processing instruction target.

`data` - The remainder of the processing instruction

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[DocumentHandler.processingInstruction\(java.lang.String, java.lang.String\)](#)

[Overview](#) [Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Interface XMLReader

All Known Subinterfaces:

[XMLFilter](#)

All Known Implementing Classes:

[ParserAdapter](#)

public abstract interface **XMLReader**

Interface for reading an XML document using callbacks.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

Note: despite its name, this interface does *not* extend the standard Java Reader interface, because reading XML is a fundamentally different activity than reading character data.

XMLReader is the interface that an XML parser's SAX2 driver must implement. This interface allows an application to set and query features and properties in the parser, to register event handlers for document processing, and to initiate a document parse.

All SAX interfaces are assumed to be synchronous: the [parse](#) methods must not return until parsing is complete, and readers must wait for an event-handler callback to return before reporting the next event.

This interface replaces the (now deprecated) SAX 1.0 [Parser](#) interface. The XMLReader interface contains two important enhancements over the old Parser interface:

1. it adds a standard way to query and set features and properties; and
2. it adds Namespace support, which is required for many higher-level XML standards.

There are adapters available to convert a SAX1 Parser to a SAX2 XMLReader and vice-versa.

Since:

SAX 2.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[XMLFilter](#), [ParserAdapter](#), [XMLReaderAdapter](#)

Method Summary

ContentHandler	getContentHandler () Return the current content handler.
DTDHandler	getDTDHandler () Return the current DTD handler.
EntityResolver	getEntityResolver () Return the current entity resolver.
ErrorHandler	getErrorHandler () Return the current error handler.
boolean	getFeature (java.lang.String name) Look up the value of a feature.
java.lang.Object	getProperty (java.lang.String name) Look up the value of a property.
void	parse (InputSource input) Parse an XML document.
void	parse (java.lang.String systemId) Parse an XML document from a system identifier (URI).
void	setContentHandler (ContentHandler handler) Allow an application to register a content event handler.
void	setDTDHandler (DTDHandler handler) Allow an application to register a DTD event handler.
void	setEntityResolver (EntityResolver resolver) Allow an application to register an entity resolver.
void	setErrorHandler (ErrorHandler handler) Allow an application to register an error event handler.
void	setFeature (java.lang.String name, boolean value) Set the state of a feature.
void	setProperty (java.lang.String name, java.lang.Object value) Set the value of a property.

Method Detail

getFeature

```
public boolean getFeature(java.lang.String name)
                        throws SAXNotRecognizedException,
                        SAXNotSupportedException
```

Look up the value of a feature.

The feature name is any fully-qualified URI. It is possible for an XMLReader to recognize a feature name but to be unable to return its value; this is especially true in the case of an adapter for a SAX1 Parser, which has no way of knowing whether the underlying parser is performing validation or expanding external entities.

All XMLReaders are required to recognize the <http://xml.org/sax/features/namespaces> and the <http://xml.org/sax/features/namespace-prefixes> feature names.

Some feature values may be available only in specific contexts, such as before, during, or after a parse.

Typical usage is something like this:

```
XMLReader r = new MySAXDriver();

                                // try to activate validation
try {
    r.setFeature("http://xml.org/sax/features/validation", true);
} catch (SAXException e) {
    System.err.println("Cannot activate validation.");
}

                                // register event handlers
r.setContentHandler(new MyContentHandler());
r.setErrorHandler(new MyErrorHandler());

                                // parse the first document
try {
    r.parse("http://www.foo.com/mydoc.xml");
} catch (IOException e) {
    System.err.println("I/O exception reading XML document");
} catch (SAXException e) {
    System.err.println("XML exception reading document.");
}
```

Implementors are free (and encouraged) to invent their own features, using names built on their own URIs.

Parameters:

name - The feature name, which is a fully-qualified URI.

Returns:

The current state of the feature (true or false).

Throws:

[SAXNotRecognizedException](#) - When the XMLReader does not recognize the feature name.

[SAXNotSupportedException](#) - When the XMLReader recognizes the feature name but cannot determine its value at this time.

See Also:

[setFeature\(java.lang.String, boolean\)](#)

setFeature

```
public void setFeature(java.lang.String name,  
                       boolean value)  
    throws SAXNotRecognizedException,  
           SAXNotSupportedException
```

Set the state of a feature.

The feature name is any fully-qualified URI. It is possible for an XMLReader to recognize a feature name but to be unable to set its value; this is especially true in the case of an adapter for a SAX1 [Parser](#), which has no way of affecting whether the underlying parser is validating, for example.

All XMLReaders are required to support setting <http://xml.org/sax/features/namespaces> to true and <http://xml.org/sax/features/namespace-prefixes> to false.

Some feature values may be immutable or mutable only in specific contexts, such as before, during, or after a parse.

Parameters:

name - The feature name, which is a fully-qualified URI.

state - The requested state of the feature (true or false).

Throws:

[SAXNotRecognizedException](#) - When the XMLReader does not recognize the feature name.

[SAXNotSupportedException](#) - When the XMLReader recognizes the feature name but cannot set the requested value.

See Also:

[getFeature\(java.lang.String\)](#)

getProperty

```
public java.lang.Object getProperty(java.lang.String name)
                               throws SAXNotRecognizedException,
                                       SAXNotSupportedException
```

Look up the value of a property.

The property name is any fully-qualified URI. It is possible for an XMLReader to recognize a property name but to be unable to return its state; this is especially true in the case of an adapter for a SAX1 [Parser](#).

XMLReaders are not required to recognize any specific property names, though an initial core set is documented for SAX2.

Some property values may be available only in specific contexts, such as before, during, or after a parse.

Implementors are free (and encouraged) to invent their own properties, using names built on their own URIs.

Parameters:

name - The property name, which is a fully-qualified URI.

Returns:

The current value of the property.

Throws:

[SAXNotRecognizedException](#) - When the XMLReader does not recognize the property name.

[SAXNotSupportedException](#) - When the XMLReader recognizes the property name but cannot determine its value at this time.

See Also:

[setProperty\(java.lang.String, java.lang.Object\)](#)

setProperty

```
public void setProperty(java.lang.String name,
                        java.lang.Object value)
                        throws SAXNotRecognizedException,
                              SAXNotSupportedException
```

Set the value of a property.

The property name is any fully-qualified URI. It is possible for an XMLReader to recognize a

property name but to be unable to set its value; this is especially true in the case of an adapter for a SAX1 [Parser](#).

XMLReaders are not required to recognize setting any specific property names, though a core set is provided with SAX2.

Some property values may be immutable or mutable only in specific contexts, such as before, during, or after a parse.

This method is also the standard mechanism for setting extended handlers.

Parameters:

`name` - The property name, which is a fully-qualified URI.

`state` - The requested value for the property.

Throws:

[SAXNotRecognizedException](#) - When the XMLReader does not recognize the property name.

[SAXNotSupportedException](#) - When the XMLReader recognizes the property name but cannot set the requested value.

setEntityResolver

```
public void setEntityResolver(EntityResolver resolver)
```

Allow an application to register an entity resolver.

If the application does not register an entity resolver, the XMLReader will perform its own default resolution.

Applications may register a new or different resolver in the middle of a parse, and the SAX parser must begin using the new resolver immediately.

Parameters:

`resolver` - The entity resolver.

Throws:

`java.lang.NullPointerException` - If the resolver argument is null.

See Also:

[getEntityResolver\(\)](#)

getEntityResolver

```
public EntityResolver getEntityResolver( )
```

Return the current entity resolver.

Returns:

The current entity resolver, or null if none has been registered.

See Also:

[setEntityResolver\(org.xml.sax.EntityResolver\)](#)

setDTDHandler

```
public void setDTDHandler(DTDHandler handler)
```

Allow an application to register a DTD event handler.

If the application does not register a DTD handler, all DTD events reported by the SAX parser will be silently ignored.

Applications may register a new or different handler in the middle of a parse, and the SAX parser must begin using the new handler immediately.

Parameters:

handler - The DTD handler.

Throws:

java.lang.NullPointerException - If the handler argument is null.

See Also:

[getDTDHandler\(\)](#)

getDTDHandler

```
public DTDHandler getDTDHandler()
```

Return the current DTD handler.

Returns:

The current DTD handler, or null if none has been registered.

See Also:

[setDTDHandler\(org.xml.sax.DTDHandler\)](#)

setContentHandler

```
public void setContentHandler(ContentHandler handler)
```

Allow an application to register a content event handler.

If the application does not register a content handler, all content events reported by the SAX parser will be silently ignored.

Applications may register a new or different handler in the middle of a parse, and the SAX parser must begin using the new handler immediately.

Parameters:

`handler` - The content handler.

Throws:

`java.lang.NullPointerException` - If the handler argument is null.

See Also:

[getContentHandler\(\)](#)

getContentHandler

```
public ContentHandler getContentHandler()
```

Return the current content handler.

Returns:

The current content handler, or null if none has been registered.

See Also:

[setContentHandler\(org.xml.sax.ContentHandler\)](#)

setErrorHandler

```
public void setErrorHandler(ErrorHandler handler)
```

Allow an application to register an error event handler.

If the application does not register an error handler, all error events reported by the SAX parser will be silently ignored; however, normal processing may not continue. It is highly recommended that all SAX applications implement an error handler to avoid unexpected bugs.

Applications may register a new or different handler in the middle of a parse, and the SAX parser must begin using the new handler immediately.

Parameters:

`handler` - The error handler.

Throws:

`java.lang.NullPointerException` - If the handler argument is null.

See Also:

[getErrorHandler\(\)](#)

getErrorHandler

```
public ErrorHandler getErrorHandler()
```

Return the current error handler.

Returns:

The current error handler, or null if none has been registered.

See Also:

[setErrorHandler\(org.xml.sax.ErrorHandler\)](#)

parse

```
public void parse(InputSource input)  
    throws java.io.IOException,  
           SAXException
```

Parse an XML document.

The application can use this method to instruct the XML reader to begin parsing an XML document from any valid input source (a character stream, a byte stream, or a URI).

Applications may not invoke this method while a parse is in progress (they should create a new XMLReader instead for each nested XML document). Once a parse is complete, an application may reuse the same XMLReader object, possibly with a different input source.

During the parse, the XMLReader will provide information about the XML document through the registered event handlers.

This method is synchronous: it will not return until parsing has ended. If a client application wants to terminate parsing early, it should throw an exception.

Parameters:

source - The input source for the top-level of the XML document.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

java.io.IOException - An IO exception from the parser, possibly from a byte stream or character stream supplied by the application.

See Also:

[InputSource](#), [parse\(java.lang.String\)](#),
[setEntityResolver\(org.xml.sax.EntityResolver\)](#),

[setDTDHandler\(org.xml.sax.DTDHandler\)](#),
[setContentHandler\(org.xml.sax.ContentHandler\)](#),
[setErrorHandler\(org.xml.sax.ErrorHandler\)](#)

parse

```
public void parse(java.lang.String systemId)  
    throws java.io.IOException,  
           SAXException
```

Parse an XML document from a system identifier (URI).

This method is a shortcut for the common case of reading a document from a system identifier. It is the exact equivalent of the following:

```
parse(new InputSource(systemId));
```

If the system identifier is a URL, it must be fully resolved by the application before it is passed to the parser.

Parameters:

`systemId` - The system identifier (URI).

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

`java.io.IOException` - An IO exception from the parser, possibly from a byte stream or character stream supplied by the application.

See Also:

[parse\(org.xml.sax.InputSource\)](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)`org.xml.sax.helpers`

Class ParserFactory

`java.lang.Object`

```
|
+--org.xml.sax.helpers.ParserFactory
```

Deprecated. *This class works with the deprecated [Parser](#) interface.*

```
public class ParserFactory
```

```
extends java.lang.Object
```

Java-specific class for dynamically loading SAX parsers.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

Note: This class is designed to work with the now-deprecated SAX1 [Parser](#) class. SAX2 applications should use [XMLReaderFactory](#) instead.

ParserFactory is not part of the platform-independent definition of SAX; it is an additional convenience class designed specifically for Java XML application writers. SAX applications can use the static methods in this class to allocate a SAX parser dynamically at run-time based either on the value of the `'org.xml.sax.parser'` system property or on a string containing the class name.

Note that the application still requires an XML parser that implements SAX1.

Since:

SAX 1.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[Parser](#), Class

Method Summary

static Parser makeParser ()	Deprecated. Create a new SAX parser using the `org.xml.sax.parser' system property.
static Parser makeParser (java.lang.String className)	Deprecated. Create a new SAX parser object using the class name provided.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

makeParser

```
public static Parser makeParser ()
    throws java.lang.ClassNotFoundException,
           java.lang.IllegalAccessException,
           java.lang.InstantiationException,
           java.lang.NullPointerException,
           java.lang.ClassCastException
```

Deprecated.

Create a new SAX parser using the `org.xml.sax.parser' system property.

The named class must exist and must implement the [Parser](#) interface.

Throws:

java.lang.NullPointerException - There is no value for the `org.xml.sax.parser' system property.

java.lang.ClassNotFoundException - The SAX parser class was not found (check your CLASSPATH).

java.lang.IllegalAccessException - The SAX parser class was found, but you do not have permission to load it.

java.lang.InstantiationException - The SAX parser class was found but could not be instantiated.

java.lang.ClassCastException - The SAX parser class was found and instantiated, but does not implement org.xml.sax.Parser.

See Also:

[makeParser \(java.lang.String\)](#), [Parser](#)

makeParser

```
public static Parser makeParser(java.lang.String className)
                                throws java.lang.ClassNotFoundException,
                                       java.lang.IllegalAccessException,
                                       java.lang.InstantiationException,
                                       java.lang.ClassCastException
```

Deprecated.

Create a new SAX parser object using the class name provided.

The named class must exist and must implement the [Parser](#) interface.

Parameters:

className - A string containing the name of the SAX parser class.

Throws:

java.lang.ClassNotFoundException - The SAX parser class was not found (check your CLASSPATH).

java.lang.IllegalAccessException - The SAX parser class was found, but you do not have permission to load it.

java.lang.InstantiationException - The SAX parser class was found but could not be instantiated.

java.lang.ClassCastException - The SAX parser class was found and instantiated, but does not implement org.xml.sax.Parser.

See Also:

[makeParser\(\)](#), [Parser](#)

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#)
[Package](#)
[Class](#)
[Use Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.parsers

Class SAXParser

java.lang.Object

|

+-- **javax.xml.parsers.SAXParser**public abstract class **SAXParser**

extends java.lang.Object

Defines the API that wraps an [XMLReader](#) implementation class. In JAXP 1.0, this class wrapped the [Parser](#) interface, however this interface was replaced by the [XMLReader](#). For ease of transition, this class continues to support the same name and interface as well as supporting new methods. An instance of this class can be obtained from the [SAXParserFactory.newSAXParser\(\)](#) method. Once an instance of this class is obtained, XML can be parsed from a variety of input sources. These input sources are InputStreams, Files, URLs, and SAX InputSources.

As the content is parsed by the underlying parser, methods of the given [HandlerBase](#) or the [DefaultHandler](#) are called.

Implementors of this class which wrap an underlying implementation can consider using the [ParserAdapter](#) class to initially adapt their SAX1 implementation to work under this revised class.

An implementation of SAXParser is *NOT* guaranteed to behave as per the specification if it is used concurrently by two or more threads. It is recommended to have one instance of the SAXParser per thread or it is upto the application to make sure about the use of SAXParser from more than one thread.

Since:

JAXP 1.0

Version:

1.0

Constructor Summary

protected	SAXParser ()
-----------	-------------------------------

Method Summary

abstract Parser	getParser () Returns the SAX parser that is encapsulated by the implementation of this class.
abstract java.lang.Object	getProperty (java.lang.String name) Returns the particular property requested for in the underlying implementation of XMLReader .
abstract XMLReader	getXMLReader () Returns the XMLReader that is encapsulated by the implementation of this class.
abstract boolean	isNamespaceAware () Indicates whether or not this parser is configured to understand namespaces.
abstract boolean	isValidating () Indicates whether or not this parser is configured to validate XML documents.
void	parse (java.io.File f, DefaultHandler dh) Parse the content of the file specified as XML using the specified DefaultHandler .
void	parse (java.io.File f, HandlerBase hb) Parse the content of the file specified as XML using the specified HandlerBase .
void	parse (InputSource is, DefaultHandler dh) Parse the content given InputSource as XML using the specified DefaultHandler .
void	parse (InputSource is, HandlerBase hb) Parse the content given InputSource as XML using the specified HandlerBase .
void	parse (java.io.InputStream is, DefaultHandler dh) Parse the content of the given InputStream instance as XML using the specified DefaultHandler .
void	parse (java.io.InputStream is, DefaultHandler dh, java.lang.String systemId) Parse the content of the given InputStream instance as XML using the specified DefaultHandler .
void	parse (java.io.InputStream is, HandlerBase hb) Parse the content of the given InputStream instance as XML using the specified HandlerBase .

void	<code>parse</code> (java.io.InputStream is, HandlerBase hb, java.lang.String systemId) Parse the content of the given InputStream instance as XML using the specified HandlerBase .
void	<code>parse</code> (java.lang.String uri, DefaultHandler dh) Parse the content described by the giving Uniform Resource Identifier (URI) as XML using the specified DefaultHandler .
void	<code>parse</code> (java.lang.String uri, HandlerBase hb) Parse the content described by the giving Uniform Resource Identifier (URI) as XML using the specified HandlerBase .
abstract void	<code>setProperty</code> (java.lang.String name, java.lang.Object value) Sets the particular property in the underlying implementation of XMLReader .

Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Constructor Detail

SAXParser

protected **SAXParser**()

Method Detail

parse

```
public void parse(java.io.InputStream is,
                  HandlerBase hb)
    throws SAXException,
           java.io.IOException
```

Parse the content of the given InputStream instance as XML using the specified [HandlerBase](#). *Use of the DefaultHandler version of this method is recommended as the HandlerBase class has been deprecated in SAX 2.0*

Parameters:

`is` - `InputStream` containing the content to be parsed.

`hb` - The `SAX HandlerBase` to use.

Throws:

`java.io.IOException` - If any IO errors occur.

`java.lang.IllegalArgumentException` - If the given `InputStream` is null.

[SAXException](#) - If the underlying parser throws a `SAXException` while parsing.

See Also:

[DocumentHandler](#)

parse

```
public void parse(java.io.InputStream is,
                  HandlerBase hb,
                  java.lang.String systemId)
    throws SAXException,
           java.io.IOException
```

Parse the content of the given `InputStream` instance as XML using the specified [HandlerBase](#). *Use of the `DefaultHandler` version of this method is recommended as the `HandlerBase` class has been deprecated in SAX 2.0*

Parameters:

`is` - `InputStream` containing the content to be parsed.

`hb` - The `SAX HandlerBase` to use.

`systemId` - The `systemId` which is needed for resolving relative URIs.

Throws:

`java.io.IOException` - If any IO errors occur.

`java.lang.IllegalArgumentException` - If the given `InputStream` is null.

[SAXException](#) - If the underlying parser throws a `SAXException` while parsing.

See Also:

[version of this method instead.](#)

parse

```
public void parse(java.io.InputStream is,
                  DefaultHandler dh)
    throws SAXException,
```

`java.io.IOException`

Parse the content of the given `InputStream` instance as XML using the specified [DefaultHandler](#).

Parameters:

`is` - `InputStream` containing the content to be parsed.

`dh` - The SAX `DefaultHandler` to use.

Throws:

`java.io.IOException` - If any IO errors occur.

`java.lang.IllegalArgumentException` - If the given `InputStream` is null.

[SAXException](#) - If the underlying parser throws a `SAXException` while parsing.

See Also:

[DocumentHandler](#)

parse

```
public void parse(java.io.InputStream is,  
                  DefaultHandler dh,  
                  java.lang.String systemId)  
    throws SAXException,  
           java.io.IOException
```

Parse the content of the given `InputStream` instance as XML using the specified [DefaultHandler](#).

Parameters:

`is` - `InputStream` containing the content to be parsed.

`dh` - The SAX `DefaultHandler` to use.

`systemId` - The `systemId` which is needed for resolving relative URIs.

Throws:

`java.io.IOException` - If any IO errors occur.

`java.lang.IllegalArgumentException` - If the given `InputStream` is null.

[SAXException](#) - If the underlying parser throws a `SAXException` while parsing.

See Also:

[version of this method instead.](#)

parse

```
public void parse(java.lang.String uri,
                  HandlerBase hb)
    throws SAXException,
           java.io.IOException
```

Parse the content described by the giving Uniform Resource Identifier (URI) as XML using the specified [HandlerBase](#). *Use of the `DefaultHandler` version of this method is recommended as the `HandlerBase` class has been deprecated in SAX 2.0*

Parameters:

`uri` - The location of the content to be parsed.

`hb` - The SAX `HandlerBase` to use.

Throws:

`java.io.IOException` - If any IO errors occur.

`java.lang.IllegalArgumentException` - If the `uri` is null.

[SAXException](#) - If the underlying parser throws a `SAXException` while parsing.

See Also:

[DocumentHandler](#)

parse

```
public void parse(java.lang.String uri,
                  DefaultHandler dh)
    throws SAXException,
           java.io.IOException
```

Parse the content described by the giving Uniform Resource Identifier (URI) as XML using the specified [DefaultHandler](#).

Parameters:

`uri` - The location of the content to be parsed.

`dh` - The SAX `DefaultHandler` to use.

Throws:

`java.io.IOException` - If any IO errors occur.

`java.lang.IllegalArgumentException` - If the `uri` is null.

[SAXException](#) - If the underlying parser throws a `SAXException` while parsing.

See Also:

[DocumentHandler](#)

parse

```
public void parse(java.io.File f,
                  HandlerBase hb)
    throws SAXException,
           java.io.IOException
```

Parse the content of the file specified as XML using the specified [HandlerBase](#). *Use of the `DefaultHandler` version of this method is recommended as the `HandlerBase` class has been deprecated in SAX 2.0*

Parameters:

f - The file containing the XML to parse

hb - The SAX `HandlerBase` to use.

Throws:

`java.io.IOException` - If any IO errors occur.

`java.lang.IllegalArgumentException` - If the File object is null.

[SAXException](#) - If the underlying parser throws a `SAXException` while parsing.

See Also:

[DocumentHandler](#)

parse

```
public void parse(java.io.File f,
                  DefaultHandler dh)
    throws SAXException,
           java.io.IOException
```

Parse the content of the file specified as XML using the specified [DefaultHandler](#).

Parameters:

f - The file containing the XML to parse

dh - The SAX `DefaultHandler` to use.

Throws:

`java.io.IOException` - If any IO errors occur.

`java.lang.IllegalArgumentException` - If the File object is null.

[SAXException](#) - If the underlying parser throws a `SAXException` while parsing.

See Also:

[DocumentHandler](#)

parse

```
public void parse(InputSource is,
                  HandlerBase hb)
    throws SAXException,
           java.io.IOException
```

Parse the content given [InputSource](#) as XML using the specified [HandlerBase](#). *Use of the `DefaultHandler` version of this method is recommended as the `HandlerBase` class has been deprecated in SAX 2.0*

Parameters:

`is` - The `InputSource` containing the content to be parsed.

`hb` - The SAX `HandlerBase` to use.

Throws:

`java.io.IOException` - If any IO errors occur.

`java.lang.IllegalArgumentException` - If the `InputSource` is null.

[SAXException](#) - If the underlying parser throws a `SAXException` while parsing.

See Also:

[DocumentHandler](#)

parse

```
public void parse(InputSource is,
                  DefaultHandler dh)
    throws SAXException,
           java.io.IOException
```

Parse the content given [InputSource](#) as XML using the specified [DefaultHandler](#).

Parameters:

`is` - The `InputSource` containing the content to be parsed.

`dh` - The SAX `DefaultHandler` to use.

Throws:

`java.io.IOException` - If any IO errors occur.

`java.lang.IllegalArgumentException` - If the `InputSource` is null.

[SAXException](#) - If the underlying parser throws a `SAXException` while parsing.

See Also:

[DocumentHandler](#)

getParser

```
public abstract Parser getParser()  
                                throws SAXException
```

Returns the SAX parser that is encapsulated by the implementation of this class.

Returns:

The SAX parser that is encapsulated by the implementation of this class.

getXMLReader

```
public abstract XMLReader getXMLReader()  
                                throws SAXException
```

Returns the [XMLReader](#) that is encapsulated by the implementation of this class.

Returns:

The XMLReader that is encapsulated by the implementation of this class.

isNamespaceAware

```
public abstract boolean isNamespaceAware()
```

Indicates whether or not this parser is configured to understand namespaces.

Returns:

true if this parser is configured to understand namespaces; false otherwise.

isValidating

```
public abstract boolean isValidating()
```

Indicates whether or not this parser is configured to validate XML documents.

Returns:

true if this parser is configured to validate XML documents; false otherwise.

setProperty

```
public abstract void setProperty(java.lang.String name,
                                   java.lang.Object value)
                                   throws SAXNotRecognizedException,
                                   SAXNotSupportedException
```

Sets the particular property in the underlying implementation of [XMLReader](#). A list of the core features and properties can be found at <http://www.megginson.com/SAX/Java/features.html>

Parameters:

name - The name of the property to be set.

value - The value of the property to be set.

Throws:

[SAXNotRecognizedException](#) - When the underlying XMLReader does not recognize the property name.

[SAXNotSupportedException](#) - When the underlying XMLReader recognizes the property name but doesn't support the property.

See Also:

[XMLReader.setProperty\(java.lang.String, java.lang.Object\)](#)

getProperty

```
public abstract java.lang.Object getProperty(java.lang.String name)
                                   throws SAXNotRecognizedException,
                                   SAXNotSupportedException
```

Returns the particular property requested for in the underlying implementation of [XMLReader](#).

Parameters:

name - The name of the property to be retrieved.

Returns:

Value of the requested property.

Throws:

[SAXNotRecognizedException](#) - When the underlying XMLReader does not recognize the property name.

[SAXNotSupportedException](#) - When the underlying XMLReader recognizes the property name but doesn't support the property.

See Also:

[XMLReader.getProperty\(java.lang.String\)](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.parsers

Class SAXParserFactory

java.lang.Object

```

|
+-- javax.xml.parsers.SAXParserFactory

```

public abstract class **SAXParserFactory**

extends java.lang.Object

Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents.

An implementation of the SAXParserFactory class is *NOT* guaranteed to be thread safe. It is up to the user application to make sure about the use of the SAXParserFactory from more than one thread. Alternatively the application can have one instance of the SAXParserFactory per thread. An application can use the same instance of the factory to obtain one or more instances of the SAXParser provided the instance of the factory isn't being used in more than one thread at a time.

The static newInstance method returns a new concrete implementation of this class.

Since:

JAXP 1.0

Version:

1.0

Constructor Summary

protected	SAXParserFactory ()
-----------	--------------------------------------

Method Summary

abstract boolean	getFeature (java.lang.String name) Returns the particular property requested for in the underlying implementation of org.xml.sax.XMLReader.
--------------------	--

boolean	<u>isNamespaceAware</u> () Indicates whether or not the factory is configured to produce parsers which are namespace aware.
boolean	<u>isValidating</u> () Indicates whether or not the factory is configured to produce parsers which validate the XML content during parse.
static <u>SAXParserFactory</u>	<u>newInstance</u> () Obtain a new instance of a SAXParserFactory.
abstract <u>SAXParser</u>	<u>newSAXParser</u> () Creates a new instance of a SAXParser using the currently configured factory parameters.
abstract void	<u>setFeature</u> (java.lang.String name, boolean value) Sets the particular feature in the underlying implementation of org.xml.sax.XMLReader.
void	<u>setNamespaceAware</u> (boolean awareness) Specifies that the parser produced by this code will provide support for XML namespaces.
void	<u>setValidating</u> (boolean validating) Specifies that the parser produced by this code will validate documents as they are parsed.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SAXParserFactory

protected **SAXParserFactory**()

Method Detail

newInstance

```
public static SAXParserFactory newInstance()
                                   throws FactoryConfigurationError
```

Obtain a new instance of a SAXParserFactory. This static method creates a new factory instance. This method uses the following ordered lookup procedure to determine the SAXParserFactory implementation class to load:

- Use the `javax.xml.parsers.SAXParserFactory` system property.
- Use the properties file "lib/jaxp.properties" in the JRE directory. This configuration file is in standard `java.util.Properties` format and contains the fully qualified name of the implementation class with the key being the system property defined above.
- Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file `META-INF/services/javax.xml.parsers.SAXParserFactory` in jars available to the runtime.
- Platform default SAXParserFactory instance.

Once an application has obtained a reference to a SAXParserFactory it can use the factory to configure and obtain parser instances.

Returns:

A new instance of a SAXParserFactory.

Throws:

[FactoryConfigurationError](#) - if the implementation is not available or cannot be instantiated.

newSAXParser

```
public abstract SAXParser newSAXParser()
                                   throws ParserConfigurationException,
                                           SAXException
```

Creates a new instance of a SAXParser using the currently configured factory parameters.

Returns:

A new instance of a SAXParser.

Throws:

[ParserConfigurationException](#) - if a parser cannot be created which satisfies the requested configuration.

setNamespaceAware

```
public void setNamespaceAware(boolean awareness)
```

Specifies that the parser produced by this code will provide support for XML namespaces. By default the value of this is set to `false`.

Parameters:

`awareness` - true if the parser produced by this code will provide support for XML namespaces; false otherwise.

setValidating

```
public void setValidating(boolean validating)
```

Specifies that the parser produced by this code will validate documents as they are parsed. By default the value of this is set to `false`.

Parameters:

`validating` - true if the parser produced by this code will validate documents as they are parsed; false otherwise.

isNamespaceAware

```
public boolean isNamespaceAware()
```

Indicates whether or not the factory is configured to produce parsers which are namespace aware.

Returns:

true if the factory is configured to produce parsers which are namespace aware; false otherwise.

isValidating

```
public boolean isValidating()
```

Indicates whether or not the factory is configured to produce parsers which validate the XML content during parse.

Returns:

true if the factory is configured to produce parsers which validate the XML content during parse; false otherwise.

setFeature

```
public abstract void setFeature(java.lang.String name,
                                boolean value)
                                throws ParserConfigurationException,
                                SAXNotRecognizedException,
                                SAXNotSupportedException
```

Sets the particular feature in the underlying implementation of org.xml.sax.XMLReader. A list of the core features and properties can be found at

<http://www.megginson.com/SAX/Java/features.html>

Parameters:

name - The name of the feature to be set.

value - The value of the feature to be set.

Throws:

[SAXNotRecognizedException](#) - When the underlying XMLReader does not recognize the property name.

[SAXNotSupportedException](#) - When the underlying XMLReader recognizes the property name but doesn't support the property.

See Also:

[XMLReader.setFeature\(java.lang.String, boolean\)](#)

getFeature

```
public abstract boolean getFeature(java.lang.String name)
                                throws ParserConfigurationException,
                                SAXNotRecognizedException,
                                SAXNotSupportedException
```

Returns the particular property requested for in the underlying implementation of org.xml.sax.XMLReader.

Parameters:

name - The name of the property to be retrieved.

Returns:

Value of the requested property.

Throws:

[SAXNotRecognizedException](#) - When the underlying XMLReader does not recognize the property name.

[SAXNotSupportedException](#) - When the underlying XMLReader recognizes the property

name but doesn't support the property.

See Also:

[XMLReader.getProperty\(java.lang.String\)](#)

[Overview](#) [Package](#) **[Class](#)** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
[SUMMARY: INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
[DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform.sax

Class SAXResult

java.lang.Object

```

|
+-- javax.xml.transform.sax.SAXResult

```

public class **SAXResult**

extends java.lang.Object

implements [Result](#)

Acts as an holder for a transformation Result.

Field Summary

static java.lang.String	FEATURE If TransformerFactory.getFeature(java.lang.String) returns true when passed this value as an argument, the Transformer supports Result output of this type.
-------------------------	--

Constructor Summary

SAXResult () Zero-argument default constructor.
SAXResult (ContentHandler handler) Create a SAXResult that targets a SAX2 ContentHandler .

Method Summary

ContentHandler	getHandler () Get the ContentHandler that is the Result.
LexicalHandler	getLexicalHandler () Get a SAX2 LexicalHandler for the output.

java.lang.String	getSystemId () Get the system identifier that was set with setSystemId.
void	setHandler (ContentHandler handler) Set the target to be a SAX2 ContentHandler .
void	setLexicalHandler (LexicalHandler handler) Set the SAX2 LexicalHandler for the output.
void	setSystemId (java.lang.String systemId) Method setSystemId Set the systemID that may be used in association with the ContentHandler .

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

FEATURE

```
public static final java.lang.String FEATURE
```

If [TransformerFactory.getFeature\(java.lang.String\)](#) returns true when passed this value as an argument, the Transformer supports Result output of this type.

Constructor Detail

SAXResult

```
public SAXResult()
```

Zero-argument default constructor.

SAXResult

```
public SAXResult(ContentHandler handler)
```

Create a SAXResult that targets a SAX2 [ContentHandler](#).

Parameters:

handler - Must be a non-null ContentHandler reference.

Method Detail

setHandler

```
public void setHandler(ContentHandler handler)
```

Set the target to be a SAX2 [ContentHandler](#).

Parameters:

handler - Must be a non-null ContentHandler reference.

getHandler

```
public ContentHandler getHandler()
```

Get the [ContentHandler](#) that is the Result.

Returns:

The ContentHandler that is to be transformation output.

setLexicalHandler

```
public void setLexicalHandler(LexicalHandler handler)
```

Set the SAX2 [LexicalHandler](#) for the output.

This is needed to handle XML comments and the like. If the lexical handler is not set, an attempt should be made by the transformer to cast the [ContentHandler](#) to a LexicalHandler.

Parameters:

handler - A non-null LexicalHandler for handling lexical parse events.

getLexicalHandler

```
public LexicalHandler getLexicalHandler()
```

Get a SAX2 [LexicalHandler](#) for the output.

Returns:

A LexicalHandler, or null.

setSystemId

```
public void setSystemId(java.lang.String systemId)
```

Method setSystemId Set the systemID that may be used in association with the [ContentHandler](#).

Specified by:

[setSystemId](#) in interface [Result](#)

Parameters:

systemId - The system identifier as a URI string.

getSystemId

```
public java.lang.String getSystemId()
```

Get the system identifier that was set with setSystemId.

Specified by:

[getSystemId](#) in interface [Result](#)

Returns:

The system identifier that was set with setSystemId, or null if setSystemId was not called.

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform.sax

Class SAXSource

java.lang.Object

```

|
+-- javax.xml.transform.sax.SAXSource

```

public class **SAXSource**

extends java.lang.Object

implements [Source](#)

Acts as an holder for SAX-style Source.

Field Summary

static java.lang.String	FEATURE If TransformerFactory.getFeature(java.lang.String) returns true when passed this value as an argument, the Transformer supports Source input of this type.
-------------------------	---

Constructor Summary

SAXSource ()	Zero-argument default constructor.
SAXSource (InputSource inputSource)	Create a SAXSource, using a SAX InputSource.
SAXSource (XMLReader reader, InputSource inputSource)	Create a SAXSource, using an XMLReader and a SAX InputSource.

Method Summary

InputSource	getInputSource () Get the SAX InputSource to be used for the Source.
-----------------------------	--

java.lang.String	<code>getSystemId</code> () Get the base ID (URI or system ID) from where URIs will be resolved.
<code>XMLReader</code>	<code>getXMLReader</code> () Get the XMLReader to be used for the Source.
void	<code>setInputSource</code> (<code>InputSource</code> inputSource) Set the SAX InputSource to be used for the Source.
void	<code>setSystemId</code> (java.lang.String systemId) Set the system identifier for this Source.
void	<code>setXMLReader</code> (<code>XMLReader</code> reader) Set the XMLReader to be used for the Source.
static <code>InputSource</code>	<code>sourceToInputSource</code> (<code>Source</code> source) Attempt to obtain a SAX InputSource object from a TrAX Source object.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

FEATURE

public static final java.lang.String **FEATURE**

If [`TransformerFactory.getFeature\(java.lang.String\)`](#) returns true when passed this value as an argument, the Transformer supports Source input of this type.

Constructor Detail

SAXSource

public **SAXSource**()

Zero-argument default constructor. If this constructor is used, and no other method is called, the [`Transformer`](#) assumes an empty input tree, with a default root node.

SAXSource

```
public SAXSource(XMLReader reader,
                 InputSource inputSource)
```

Create a SAXSource, using an [XMLReader](#) and a SAX InputSource. The [Transformer](#) or [SAXTransformerFactory](#) will set itself to be the reader's [ContentHandler](#), and then will call reader.parse(inputSource).

Parameters:

reader - An XMLReader to be used for the parse.

inputSource - A SAX input source reference that must be non-null and that will be passed to the reader parse method.

SAXSource

```
public SAXSource(InputSource inputSource)
```

Create a SAXSource, using a SAX InputSource. The [Transformer](#) or [SAXTransformerFactory](#) creates a reader via [XMLReaderFactory](#) (if setXMLReader is not used), sets itself as the reader's [ContentHandler](#), and calls reader.parse(inputSource).

Parameters:

inputSource - An input source reference that must be non-null and that will be passed to the parse method of the reader.

Method Detail

setXMLReader

```
public void setXMLReader(XMLReader reader)
```

Set the XMLReader to be used for the Source.

Parameters:

reader - A valid XMLReader or XMLFilter reference.

getXMLReader

```
public XMLReader getXMLReader()
```

Get the XMLReader to be used for the Source.

Returns:

A valid XMLReader or XMLFilter reference, or null.

setInputSource

```
public void setInputSource(InputSource inputSource)
```

Set the SAX InputSource to be used for the Source.

Parameters:

`inputSource` - A valid InputSource reference.

getInputSource

```
public InputSource getInputSource()
```

Get the SAX InputSource to be used for the Source.

Returns:

A valid InputSource reference, or null.

setSystemId

```
public void setSystemId(java.lang.String systemId)
```

Set the system identifier for this Source. If an input source has already been set, it will set the system ID or that input source, otherwise it will create a new input source.

The system identifier is optional if there is a byte stream or a character stream, but it is still useful to provide one, since the application can use it to resolve relative URIs and can include it in error messages and warnings (the parser will attempt to open a connection to the URI only if no byte stream or character stream is specified).

Specified by:

[setSystemId](#) in interface [Source](#)

Parameters:

`systemId` - The system identifier as a URI string.

getSystemId

```
public java.lang.String getSystemId()
```

Get the base ID (URI or system ID) from where URIs will be resolved.

Specified by:

[getSystemId](#) in interface [Source](#)

Returns:

Base URL for the Source, or null.

sourceToInputSource

```
public static InputSource sourceToInputSource(Source source)
```

Attempt to obtain a SAX InputSource object from a TrAX Source object.

Parameters:

`source` - Must be a non-null Source reference.

Returns:

An InputSource, or null if Source can not be converted.

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform.stream

Class StreamResult

java.lang.Object

|

+-- **javax.xml.transform.stream.StreamResult**public class **StreamResult**

extends java.lang.Object

implements [Result](#)

Acts as an holder for a transformation result, which may be XML, plain Text, HTML, or some other form of markup.

Field Summary

static java.lang.String	FEATURE
	If TransformerFactory.getFeature(java.lang.String) returns true when passed this value as an argument, the Transformer supports Result output of this type.

Constructor Summary

[**StreamResult**](#)()

Zero-argument default constructor.

[**StreamResult**](#)(java.io.File f)

Construct a StreamResult from a File.

[**StreamResult**](#)(java.io.OutputStream outputStream)

Construct a StreamResult from a byte stream.

[**StreamResult**](#)(java.lang.String systemId)

Construct a StreamResult from a URL.

[**StreamResult**](#)(java.io.Writer writer)

Construct a StreamResult from a character stream.

Method Summary

<code>java.io.OutputStream</code>	<code>getOutputStream()</code> Get the byte stream that was set with <code>setOutputStream</code> .
<code>java.lang.String</code>	<code>getSystemId()</code> Get the system identifier that was set with <code>setSystemId</code> .
<code>java.io.Writer</code>	<code>getWriter()</code> Get the character stream that was set with <code>setWriter</code> .
<code>void</code>	<code>setOutputStream(java.io.OutputStream outputStream)</code> Set the <code>ByteStream</code> that is to be written to.
<code>void</code>	<code>setSystemId(java.io.File f)</code> Set the system ID from a <code>File</code> reference.
<code>void</code>	<code>setSystemId(java.lang.String systemId)</code> Set the <code>systemID</code> that may be used in association with the byte or character stream, or, if neither is set, use this value as a writeable URI (probably a file name).
<code>void</code>	<code>setWriter(java.io.Writer writer)</code> Set the writer that is to receive the result.

Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Field Detail

FEATURE

`public static final java.lang.String` **FEATURE**

If [`TransformerFactory.getFeature\(java.lang.String\)`](#) returns true when passed this value as an argument, the Transformer supports Result output of this type.

Constructor Detail

StreamResult

```
public StreamResult()
```

Zero-argument default constructor.

StreamResult

```
public StreamResult(java.io.OutputStream outputStream)
```

Construct a StreamResult from a byte stream. Normally, a stream should be used rather than a reader, so that the transformer may use instructions contained in the transformation instructions to control the encoding.

Parameters:

outputStream - A valid OutputStream reference.

StreamResult

```
public StreamResult(java.io.Writer writer)
```

Construct a StreamResult from a character stream. Normally, a stream should be used rather than a reader, so that the transformer may use instructions contained in the transformation instructions to control the encoding. However, there are times when it is useful to write to a character stream, such as when using a StringWriter.

Parameters:

writer - A valid Writer reference.

StreamResult

```
public StreamResult(java.lang.String systemId)
```

Construct a StreamResult from a URL.

Parameters:

systemId - Must be a String that conforms to the URI syntax.

StreamResult

```
public StreamResult(java.io.File f)
```

Construct a StreamResult from a File.

Parameters:

f - Must a non-null File reference.

Method Detail

setOutputStream

```
public void setOutputStream(java.io.OutputStream outputStream)
```

Set the ByteStream that is to be written to. Normally, a stream should be used rather than a reader, so that the transformer may use instructions contained in the transformation instructions to control the encoding.

Parameters:

outputStream - A valid OutputStream reference.

getOutputStream

```
public java.io.OutputStream getOutputStream()
```

Get the byte stream that was set with setOutputStream.

Returns:

The byte stream that was set with setOutputStream, or null if setOutputStream or the ByteStream constructor was not called.

setWriter

```
public void setWriter(java.io.Writer writer)
```

Set the writer that is to receive the result. Normally, a stream should be used rather than a writer, so that the transformer may use instructions contained in the transformation instructions to control the encoding. However, there are times when it is useful to write to a writer, such as when using a StringWriter.

Parameters:

writer - A valid Writer reference.

getWriter

```
public java.io.Writer getWriter()
```

Get the character stream that was set with setWriter.

Returns:

The character stream that was set with setWriter, or null if setWriter or the Writer constructor was not called.

setSystemId

```
public void setSystemId(java.lang.String systemId)
```

Set the systemID that may be used in association with the byte or character stream, or, if neither is set, use this value as a writeable URI (probably a file name).

Specified by:

[setSystemId](#) in interface [Result](#)

Parameters:

systemId - The system identifier as a URI string.

setSystemId

```
public void setSystemId(java.io.File f)
```

Set the system ID from a File reference.

Parameters:

f - Must a non-null File reference.

getSystemId

```
public java.lang.String getSystemId()
```

Get the system identifier that was set with setSystemId.

Specified by:

[getSystemId](#) in interface [Result](#)

Returns:

The system identifier that was set with setSystemId, or null if setSystemId was not called.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: INNER | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV PACKAGE](#)
[NEXT PACKAGE](#)
[FRAMES](#)
[NO FRAMES](#)

Package javax.xml.parsers

Provides classes allowing the processing of XML documents.

See:

[Description](#)

Class Summary

DocumentBuilder	Defines the API to obtain DOM Document instances from an XML document.
DocumentBuilderFactory	Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.
SAXParser	Defines the API that wraps an XMLReader implementation class.
SAXParserFactory	Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents. An implementation of the SAXParserFactory class is <i>NOT</i> guaranteed to be thread safe.

Exception Summary

ParserConfigurationException	Indicates a serious configuration error.
--	--

Error Summary

FactoryConfigurationError	Thrown when a problem with configuration with the Parser Factories exists.
---	--

Package javax.xml.parsers Description

Provides classes allowing the processing of XML documents. Two types of plugable parsers are supported:

- SAX (Simple API for XML)
- DOM (Document Object Model)

[Overview](#) **Package** [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV PACKAGE](#) [NEXT PACKAGE](#)

[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV PACKAGE](#)
[NEXT PACKAGE](#)
[FRAMES](#)
[NO FRAMES](#)

Package javax.xml.transform

This package defines the generic APIs for processing transformation instructions, and performing a transformation from source to result.

See:

[Description](#)

Interface Summary

<i><u>ErrorListener</u></i>	To provide customized error handling, implement this interface and use the setErrorListener method to register an instance of the implementation with the Transformer .
<i><u>Result</u></i>	An object that implements this interface contains the information needed to build a transformation result tree.
<i><u>Source</u></i>	An object that implements this interface contains the information needed to act as source input (XML source or transformation instructions).
<i><u>SourceLocator</u></i>	This interface is primarily for the purposes of reporting where an error occurred in the XML source or transformation instructions.
<i><u>Templates</u></i>	An object that implements this interface is the runtime representation of processed transformation instructions.
<i><u>URIResolver</u></i>	An object that implements this interface that can be called by the processor to turn a URI used in document(), xsl:import, or xsl:include into a Source object.

Class Summary

<i><u>OutputKeys</u></i>	Provides string constants that can be used to set output properties for a Transformer, or to retrieve output properties from a Transformer or Templates object.
<i><u>Transformer</u></i>	An instance of this abstract class can transform a source tree into a result tree.
<i><u>TransformerFactory</u></i>	A TransformerFactory instance can be used to create Transformer and Templates objects.

Exception Summary

<u>TransformerConfigurationException</u>	Indicates a serious configuration error.
<u>TransformerException</u>	This class specifies an exceptional condition that occurred during the transformation process.

Error Summary

<u>TransformerFactoryConfigurationError</u>	Thrown when a problem with configuration with the Transformer Factories exists.
---	---

Package javax.xml.transform Description

This package defines the generic APIs for processing transformation instructions, and performing a transformation from source to result. These interfaces have no dependencies on SAX or the DOM standard, and try to make as few assumptions as possible about the details of the source and result of a transformation. It achieves this by defining [Source](#) and [Result](#) interfaces.

To define concrete classes for the user, the API defines specializations of the interfaces found at the root level. These interfaces are found in [javax.xml.transform.sax](#), [javax.xml.transform.dom](#), and [javax.xml.transform.stream](#).

Creating Objects

The API allows a concrete [TransformerFactory](#) object to be created from the static function [TransformerFactory.newInstance\(\)](#).

Specification of Inputs and Outputs

This API defines two interface objects called [Source](#) and [Result](#). In order to pass Source and Result objects to the interfaces, concrete classes must be used. Three concrete representations are defined for each of these objects: [StreamSource](#) and [StreamResult](#), [SAXSource](#) and [SAXResult](#), and [DOMSource](#) and [DOMResult](#). Each of these objects defines a FEATURE string (which is in the form of a URL), which can be passed into [TransformerFactory.getFeature\(java.lang.String\)](#) to see if the given type of Source or Result object is supported. For instance, to test if a DOMSource and a StreamResult is supported, you can apply the following test.

```
TransformerFactory tfactory = TransformerFactory.newInstance();

if (tfactory.getFeature(DOMSource.FEATURE) &&
    tfactory.getFeature(StreamResult.FEATURE))
```

```
{
    ...
}
```

Qualified Name Representation

[Namespaces](#) present something of a problem area when dealing with XML objects. Qualified Names appear in XML markup as prefixed names. But the prefixes themselves do not hold identity. Rather, it is the URIs that they contextually map to that hold the identity. Therefore, when passing a Qualified Name like "xyz:foo" among Java programs, one must provide a means to map "xyz" to a namespace.

One solution has been to create a "QName" object that holds the namespace URI, as well as the prefix and local name, but this is not always an optimal solution, as when, for example, you want to use unique strings as keys in a dictionary object. Not having a string representation also makes it difficult to specify a namespaced identity outside the context of an XML document.

In order to pass namespaced values to transformations, for instance as a set of properties to the Serializer, this specification defines that a String "qname" object parameter be passed as two-part string, the namespace URI enclosed in curly braces ({ }), followed by the local name. If the qname has a null URI, then the String object only contains the local name. An application can safely check for a non-null URI by testing to see if the first character of the name is a '{' character.

For example, if a URI and local name were obtained from an element defined with `<xyz:foo xmlns:xyz="http://xyz.foo.com/yada/baz.html"/>`, then the Qualified Name would be "{http://xyz.foo.com/yada/baz.html}foo". Note that the prefix is lost.

Result Tree Serialization

Serialization of the result tree to a stream can be controlled with the [Transformer.setOutputProperties\(java.util.Properties\)](#) and the [Transformer.setOutputProperty\(java.lang.String, java.lang.String\)](#) methods. Strings that match the [XSLT specification for xsl:output attributes](#) can be referenced from the [OutputKeys](#) class. Other strings can be specified as well. If the transformer does not recognize an output key, a `IllegalArgumentException` is thrown, unless the key name is [namespace qualified](#). Output key names that are qualified by a namespace are ignored or passed on to the serializer mechanism.

If all that is desired is the simple identity transformation of a source to a result, then [TransformerFactory](#) provides a [TransformerFactory.newTransformer\(\)](#) method with no arguments. This method creates a Transformer that effectively copies the source to the result. This method may be used to create a DOM from SAX events or to create an XML or HTML stream from a DOM or SAX events.

Exceptions and Error Reporting

The transformation API throw three types of specialized exceptions. A [TransformerFactoryConfigurationError](#) is parallel to the [FactoryConfigurationError](#), and is thrown when a configuration problem with the TransformerFactory exists. This error will typically be thrown when the transformation factory class specified with the "javax.xml.transform.TransformerFactory" system property cannot be found or instantiated.

A [TransformerConfigurationException](#) may be thrown if for any reason a Transformer can not be created. A TransformerConfigurationException may be thrown if there is a syntax error in the transformation instructions, for example when [TransformerFactory.newTransformer\(javax.xml.transform.Source\)](#) is called.

[TransformerException](#) is a general exception that occurs during the course of a transformation. A transformer exception may wrap another exception, and if any of the [TransformerException.printStackTrace\(\)](#) methods are called on it, it will produce a list of stack dumps, starting from the most recent. The transformer exception also provides a [SourceLocator](#) object which indicates where in the source tree or transformation instructions the error occurred. [TransformerException.getMessageAndLocation\(\)](#) may be called to get an error message with location info, and [TransformerException.getLocationAsString\(\)](#) may be called to get just the location string.

Transformation warnings and errors are normally first sent to a [ErrorListener](#), at which point the implementor may decide to report the error or warning, and may decide to throw an exception for a non-fatal error. The error listener may be set via [TransformerFactory.setErrorListener\(javax.xml.transform.ErrorListener\)](#) for reporting errors that have to do with syntax errors in the transformation instructions, or via [Transformer.setErrorListener\(javax.xml.transform.ErrorListener\)](#) to report errors that occur during the transformation. The error listener on both objects should always be valid and non-null, whether set by the user or a default implementation provided by the processor.

Resolution of URIs within a transformation

The API provides a way for URIs referenced from within the stylesheet instructions or within the transformation to be resolved by the calling application. This can be done by creating a class that implements the [URIResolver](#) interface, with its one method, [URIResolver.resolve\(java.lang.String, java.lang.String\)](#), and use this class to set the URI resolution for the transformation instructions or transformation with [TransformerFactory.setURIResolver\(javax.xml.transform.URIResolver\)](#) or [Transformer.setURIResolver\(javax.xml.transform.URIResolver\)](#). The [URIResolver.resolve](#) method takes two String arguments, the URI found in the stylesheet instructions or built as part of the transformation process, and the base URI in effect when the URI passed as the first argument was encountered. The returned [Source](#) object must be usable by the

transformer, as specified in its implemented features.

[Overview](#) **Package** [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV PACKAGE](#) [NEXT PACKAGE](#)

[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV PACKAGE](#)
[NEXT PACKAGE](#)
[FRAMES](#)
[NO FRAMES](#)

Package javax.xml.transform.dom

This package implements DOM-specific transformation APIs. The [DOMSource](#) class allows the client of the implementation of this API to specify a DOM [Node](#) as the source of the input tree.

See:

[Description](#)

Interface Summary

DOMLocator	Indicates the position of a node in a source DOM, intended primarily for error reporting.
----------------------------	---

Class Summary

DOMResult	Acts as a holder for a transformation result tree, in the form of a Document Object Model (DOM) tree.
DOMSource	Acts as a holder for a transformation Source tree in the form of a Document Object Model (DOM) tree.

Package javax.xml.transform.dom Description

This package implements DOM-specific transformation APIs.

The [DOMSource](#) class allows the client of the implementation of this API to specify a DOM [Node](#) as the source of the input tree. The model of how the Transformer deals with the DOM tree in terms of mismatches with the [XSLT data model](#) or other data models is beyond the scope of this document. Any of the nodes derived from [Node](#) are legal input.

The [DOMResult](#) class allows a [Node](#) to be specified to which result DOM nodes will be appended. If an output node is not specified, the transformer will use [DocumentBuilder.newDocument\(\)](#) to create an output [Document](#) node. If a node is specified, it should be one of the following: [Document](#), [Element](#), or [DocumentFragment](#). Specification of any other node type is implementation dependent and undefined by this API. If the result is a [Document](#), the output of the transformation must have a single element root to set as the document element.

The [DOMLocator](#) node may be passed to [TransformerException](#) objects, and retrieved by trying

to cast the result of the [TransformerException.getLocator\(\)](#) method. The implementation has no responsibility to use a [DOMLocator](#) instead of a [SourceLocator](#) (though line numbers and the like do not make much sense for a DOM), so the result of `getLocator` must always be tested with an instanceof.

[Overview](#) **Package** [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV PACKAGE](#) [NEXT PACKAGE](#)

[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface Node

All Known Subinterfaces:

[Attr](#), [CDATASection](#), [CharacterData](#), [Comment](#), [Document](#), [DocumentFragment](#), [DocumentType](#), [Element](#), [Entity](#), [EntityReference](#), [Notation](#), [ProcessingInstruction](#), [Text](#)

public abstract interface **Node**

The Node interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree. While all objects implementing the Node interface expose methods for dealing with children, not all objects implementing the Node interface may have children. For example, Text nodes may not have children, and adding children to such nodes results in a `DOMException` being raised.

The attributes `nodeName`, `nodeValue` and `attributes` are included as a mechanism to get at node information without casting down to the specific derived interface. In cases where there is no obvious mapping of these attributes for a specific `nodeType` (e.g., `nodeValue` for an `Element` or `attributes` for a `Comment`), this returns `null`. Note that the specialized interfaces may contain additional and more convenient mechanisms to get and set the relevant information.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Field Summary

static short	ATTRIBUTE_NODE The node is an <code>Attr</code> .
static short	CDATA_SECTION_NODE The node is a <code>CDATASection</code> .
static short	COMMENT_NODE The node is a <code>Comment</code> .
static short	DOCUMENT_FRAGMENT_NODE The node is a <code>DocumentFragment</code> .
static short	DOCUMENT_NODE The node is a <code>Document</code> .
static short	DOCUMENT_TYPE_NODE The node is a <code>DocumentType</code> .

static short	ELEMENT_NODE The node is an Element.
static short	ENTITY_NODE The node is an Entity.
static short	ENTITY_REFERENCE_NODE The node is an EntityReference.
static short	NOTATION_NODE The node is a Notation.
static short	PROCESSING_INSTRUCTION_NODE The node is a ProcessingInstruction.
static short	TEXT_NODE The node is a Text node.

Method Summary

Node	appendChild (Node newChild) Adds the node newChild to the end of the list of children of this node.
Node	cloneNode (boolean deep) Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes.
NamedNodeMap	getAttributes () A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.
NodeList	getChildNodes () A NodeList that contains all children of this node.
Node	getFirstChild () The first child of this node.
Node	getLastChild () The last child of this node.
java.lang.String	getLocalName () Returns the local part of the qualified name of this node.
java.lang.String	getNamespaceURI () The namespace URI of this node, or null if it is unspecified.
Node	getNextSibling () The node immediately following this node.
java.lang.String	getNodeName () The name of this node, depending on its type; see the table above.

short	<code>getNodeType()</code> A code representing the type of the underlying object, as defined above.
java.lang.String	<code>getNodeValue()</code> The value of this node, depending on its type; see the table above.
Document	<code>getOwnerDocument()</code> The Document object associated with this node.
Node	<code>getParentNode()</code> The parent of this node.
java.lang.String	<code>getPrefix()</code> The namespace prefix of this node, or null if it is unspecified.
Node	<code>getPreviousSibling()</code> The node immediately preceding this node.
boolean	<code>hasAttributes()</code> Returns whether this node (if it is an element) has any attributes.
boolean	<code>hasChildNodes()</code> Returns whether this node has any children.
Node	<code>insertBefore(Node newChild, Node refChild)</code> Inserts the node <code>newChild</code> before the existing child node <code>refChild</code> .
boolean	<code>isSupported(java.lang.String feature, java.lang.String version)</code> Tests whether the DOM implementation implements a specific feature and that feature is supported by this node.
void	<code>normalize()</code> Puts all Text nodes in the full depth of the sub-tree underneath this Node, including attribute nodes, into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates Text nodes, i.e., there are neither adjacent Text nodes nor empty Text nodes.
Node	<code>removeChild(Node oldChild)</code> Removes the child node indicated by <code>oldChild</code> from the list of children, and returns it.
Node	<code>replaceChild(Node newChild, Node oldChild)</code> Replaces the child node <code>oldChild</code> with <code>newChild</code> in the list of children, and returns the <code>oldChild</code> node.
void	<code>setNodeValue(java.lang.String nodeValue)</code>
void	<code>setPrefix(java.lang.String prefix)</code>

Field Detail

ELEMENT_NODE

```
public static final short ELEMENT_NODE
```

The node is an Element.

ATTRIBUTE_NODE

```
public static final short ATTRIBUTE_NODE
```

The node is an Attr.

TEXT_NODE

```
public static final short TEXT_NODE
```

The node is a Text node.

CDATA_SECTION_NODE

```
public static final short CDATA_SECTION_NODE
```

The node is a CDATASection.

ENTITY_REFERENCE_NODE

```
public static final short ENTITY_REFERENCE_NODE
```

The node is an EntityReference.

ENTITY_NODE

```
public static final short ENTITY_NODE
```

The node is an Entity.

PROCESSING_INSTRUCTION_NODE

```
public static final short PROCESSING_INSTRUCTION_NODE
```

The node is a `ProcessingInstruction`.

COMMENT_NODE

```
public static final short COMMENT_NODE
```

The node is a `Comment`.

DOCUMENT_NODE

```
public static final short DOCUMENT_NODE
```

The node is a `Document`.

DOCUMENT_TYPE_NODE

```
public static final short DOCUMENT_TYPE_NODE
```

The node is a `DocumentType`.

DOCUMENT_FRAGMENT_NODE

```
public static final short DOCUMENT_FRAGMENT_NODE
```

The node is a `DocumentFragment`.

NOTATION_NODE

```
public static final short NOTATION_NODE
```

The node is a `Notation`.

Method Detail

getNodeName

```
public java.lang.String getNodeName()
```

The name of this node, depending on its type; see the table above.

getNodeValue

```
public java.lang.String getNodeValue()  
                        throws DOMException
```

The value of this node, depending on its type; see the table above. When it is defined to be null, setting it has no effect.

Throws:

[DOMException](#) - NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

[DOMException](#) - DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a DOMString variable on the implementation platform.

setNodeValue

```
public void setNodeValue(java.lang.String nodeValue)  
                        throws DOMException
```

getNodeType

```
public short getNodeType()
```

A code representing the type of the underlying object, as defined above.

getParentNode

```
public Node getParentNode( )
```

The parent of this node. All nodes, except `Attr`, `Document`, `DocumentFragment`, `Entity`, and `Notation` may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is `null`.

getChildNodes

```
public NodeList getChildNodes( )
```

A `NodeList` that contains all children of this node. If there are no children, this is a `NodeList` containing no nodes.

getFirstChild

```
public Node getFirstChild( )
```

The first child of this node. If there is no such node, this returns `null`.

getLastChild

```
public Node getLastChild( )
```

The last child of this node. If there is no such node, this returns `null`.

getPreviousSibling

```
public Node getPreviousSibling( )
```

The node immediately preceding this node. If there is no such node, this returns `null`.

getNextSibling

```
public Node getNextSibling( )
```

The node immediately following this node. If there is no such node, this returns `null`.

getAttributes

```
public NamedNodeMap getAttributes()
```

A [NamedNodeMap](#) containing the attributes of this node (if it is an [Element](#)) or null otherwise.

getOwnerDocument

```
public Document getOwnerDocument()
```

The [Document](#) object associated with this node. This is also the [Document](#) object used to create new nodes. When this node is a [Document](#) or a [DocumentType](#) which is not used with any [Document](#) yet, this is null.

insertBefore

```
public Node insertBefore(Node newChild,  
                           Node refChild)  
    throws DOMException
```

Inserts the node `newChild` before the existing child node `refChild`. If `refChild` is null, insert `newChild` at the end of the list of children.

If `newChild` is a [DocumentFragment](#) object, all of its children are inserted, in the same order, before `refChild`. If the `newChild` is already in the tree, it is first removed.

Parameters:

`newChild` - node to insert.

`refChild` - reference node, i.e., the node before which the new node must be inserted.

Returns:

The node being inserted.

Throws:

[DOMException](#) - `HIERARCHY_REQUEST_ERR`: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to insert is one of this node's ancestors.

`WRONG_DOCUMENT_ERR`: Raised if `newChild` was created from a different document than the one that created this node.

`NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is readonly or if the parent of the node being inserted is readonly.

`NOT_FOUND_ERR`: Raised if `refChild` is not a child of this node.

replaceChild

```
public Node replaceChild(Node newChild,
                        Node oldChild)
    throws DOMException
```

Replaces the child node `oldChild` with `newChild` in the list of children, and returns the `oldChild` node.

If `newChild` is a `DocumentFragment` object, `oldChild` is replaced by all of the `DocumentFragment` children, which are inserted in the same order. If the `newChild` is already in the tree, it is first removed.

Parameters:

`newChild` - new node to put in the child list.

`oldChild` - node being replaced in the list.

Returns:

The node replaced.

Throws:

[DOMException](#) - `HIERARCHY_REQUEST_ERR`: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to put in is one of this node's ancestors.

`WRONG_DOCUMENT_ERR`: Raised if `newChild` was created from a different document than the one that created this node.

`NO_MODIFICATION_ALLOWED_ERR`: Raised if this node or the parent of the new node is readonly.

`NOT_FOUND_ERR`: Raised if `oldChild` is not a child of this node.

removeChild

```
public Node removeChild(Node oldChild)
    throws DOMException
```

Removes the child node indicated by `oldChild` from the list of children, and returns it.

Parameters:

`oldChild` - node being removed.

Returns:

The node removed.

Throws:

[DOMException](#) - `NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is readonly.

NOT_FOUND_ERR: Raised if `oldChild` is not a child of this node.

appendChild

```
public Node appendChild(Node newChild)
    throws DOMException
```

Adds the node `newChild` to the end of the list of children of this node. If the `newChild` is already in the tree, it is first removed.

Parameters:

`newChild` - node to add. If it is a `DocumentFragment` object, the entire contents of the document fragment are moved into the child list of this node

Returns:

The node added.

Throws:

[DOMException](#) - HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node's ancestors.

WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this node.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

hasChildNodes

```
public boolean hasChildNodes()
```

Returns whether this node has any children.

Returns:

`true` if this node has any children, `false` otherwise.

cloneNode

```
public Node cloneNode(boolean deep)
```

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent; (`parentNode` is `null`).

Cloning an `Element` copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child `Text` node. Cloning an `Attribute` directly, as opposed to be cloned as part of an `Element` cloning operation, returns a specified

attribute (specified is true). Cloning any other type of node simply returns a copy of this node.

Note that cloning an immutable subtree results in a mutable copy, but the children of an `EntityReference` clone are readonly. In addition, clones of unspecified `Attr` nodes are specified. And, cloning `Document`, `DocumentType`, `Entity`, and `Notation` nodes is implementation dependent.

Parameters:

`deepIf` - true, recursively clone the subtree under the specified node; if false, clone only the node itself (and its attributes, if it is an `Element`).

Returns:

The duplicate node.

normalize

```
public void normalize()
```

Puts all `Text` nodes in the full depth of the sub-tree underneath this `Node`, including attribute nodes, into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates `Text` nodes, i.e., there are neither adjacent `Text` nodes nor empty `Text` nodes. This can be used to ensure that the DOM view of a document is the same as if it were saved and re-loaded, and is useful when operations (such as `XPointer` lookups) that depend on a particular document tree structure are to be used. In cases where the document contains `CDATASections`, the `normalize` operation alone may not be sufficient, since `XPointers` do not differentiate between `Text` nodes and `CDATASection` nodes.

isSupported

```
public boolean isSupported(java.lang.String feature,  
                             java.lang.String version)
```

Tests whether the DOM implementation implements a specific feature and that feature is supported by this node.

Parameters:

`featureThe` - name of the feature to test. This is the same name which can be passed to the method `hasFeature` on `DOMImplementation`.

`versionThis` - is the version number of the feature to test. In Level 2, version 1, this is the string "2.0". If the version is not specified, supporting any version of the feature will cause the method to return true.

Returns:

Returns true if the specified feature is supported on this node, false otherwise.

Since:

getNamespaceURI

```
public java.lang.String getNamespaceURI()
```

The namespace URI of this node, or `null` if it is unspecified.

This is not a computed value that is the result of a namespace lookup based on an examination of the namespace declarations in scope. It is merely the namespace URI given at creation time.

For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `createElement` from the `Document` interface, this is always `null`. Per the Namespaces in XML Specification an attribute does not inherit its namespace from the element it is attached to. If an attribute is not explicitly given a namespace, it simply has no namespace.

Since:

DOM Level 2

getPrefix

```
public java.lang.String getPrefix()
```

The namespace prefix of this node, or `null` if it is unspecified.

Note that setting this attribute, when permitted, changes the `nodeName` attribute, which holds the qualified name, as well as the `tagName` and `name` attributes of the `Element` and `Attr` interfaces, when applicable.

Note also that changing the prefix of an attribute that is known to have a default value, does not make a new attribute with the default value and the original prefix appear, since the `namespaceURI` and `localName` do not change.

For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `createElement` from the `Document` interface, this is always `null`.

Throws:

[DOMException](#) - `INVALID_CHARACTER_ERR`: Raised if the specified prefix contains an illegal character.

`NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is readonly.

`NAMESPACE_ERR`: Raised if the specified `prefix` is malformed, if the `namespaceURI` of this node is `null`, if the specified `prefix` is "xml" and the `namespaceURI` of this node is different from "

`http://www.w3.org/XML/1998/namespace`", if this node is an attribute and the specified `prefix` is "xmlns" and the `namespaceURI` of this node is different from "

`http://www.w3.org/2000/xmlns/`", or if this node is an attribute and the `qualifiedName` of this node is "xmlns" .

Since:

DOM Level 2

setPrefix

```
public void setPrefix(java.lang.String prefix)
    throws DOMException
```

getLocalName

```
public java.lang.String getLocalName()
```

Returns the local part of the qualified name of this node.

For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `createElement` from the Document interface, this is always `null`.

Since:

DOM Level 2

hasAttributes

```
public boolean hasAttributes()
```

Returns whether this node (if it is an element) has any attributes.

Returns:

`true` if this node has any attributes, `false` otherwise.

Since:

DOM Level 2

Overview	Package	Class	Use	Tree	Deprecated	Index	Help
--------------------------	-------------------------	-----------------------	---------------------	----------------------	----------------------------	-----------------------	----------------------

PREV CLASS	NEXT CLASS
----------------------------	----------------------------

FRAMES	NO FRAMES
------------------------	---------------------------

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV](#) [NEXT](#)

[FRAMES](#) [NO FRAMES](#)

[A](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [L](#) [M](#) [N](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#)

A

[**addAttribute\(String, String, String\)**](#) - Method in class org.xml.sax.helpers.[AttributeListImpl](#)

Deprecated. Add an attribute to an attribute list.

[**addAttribute\(String, String, String, String, String\)**](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Add an attribute to the end of the list.

[**appendChild\(Node\)**](#) - Method in interface org.w3c.dom.[Node](#)

Adds the node `newChild` to the end of the list of children of this node.

[**appendData\(String\)**](#) - Method in interface org.w3c.dom.[CharacterData](#)

Append the string to the end of the character data of the node.

[**Attr**](#) - interface org.w3c.dom.[Attr](#).

The `Attr` interface represents an attribute in an `Element` object.

[**ATTRIBUTE_NODE**](#) - Static variable in interface org.w3c.dom.[Node](#)

The node is an `Attr`.

[**attributeDecl\(String, String, String, String, String\)**](#) - Method in interface org.xml.sax.ext.[DeclHandler](#)

Report an attribute type declaration.

[**AttributeList**](#) - interface org.xml.sax.[AttributeList](#).

Deprecated. *This interface has been replaced by the SAX2 [Attributes](#) interface, which includes Namespace support.*

[**AttributeListImpl**](#) - class org.xml.sax.helpers.[AttributeListImpl](#).

Deprecated. *This class implements a deprecated interface, [AttributeList](#); that interface has been replaced by [Attributes](#), which is implemented in the [AttributesImpl](#) helper class.*

[**AttributeListImpl\(\)**](#) - Constructor for class org.xml.sax.helpers.[AttributeListImpl](#)

Deprecated. Create an empty attribute list.

[**AttributeListImpl\(AttributeList\)**](#) - Constructor for class org.xml.sax.helpers.[AttributeListImpl](#)

Deprecated. Construct a persistent copy of an existing attribute list.

[**Attributes**](#) - interface org.xml.sax.[Attributes](#).

Interface for a list of XML attributes.

[**AttributesImpl**](#) - class org.xml.sax.helpers.[AttributesImpl](#).

Default implementation of the `Attributes` interface.

[AttributesImpl\(\)](#) - Constructor for class org.xml.sax.helpers.[AttributesImpl](#)

Construct a new, empty `AttributesImpl` object.

[AttributesImpl\(Attributes\)](#) - Constructor for class org.xml.sax.helpers.[AttributesImpl](#)

Copy an existing `Attributes` object.

C

[CDATA_SECTION_ELEMENTS](#) - Static variable in class javax.xml.transform.[OutputKeys](#)

`cdata-section-elements` = expanded names.

[CDATA_SECTION_NODE](#) - Static variable in interface org.w3c.dom.[Node](#)

The node is a `CDATASection`.

[CDATASection](#) - interface org.w3c.dom.[CDATASection](#).

CDATA sections are used to escape blocks of text containing characters that would otherwise be regarded as markup.

[CharacterData](#) - interface org.w3c.dom.[CharacterData](#).

The `CharacterData` interface extends `Node` with a set of attributes and methods for accessing character data in the DOM.

[characters\(char\[\], int, int\)](#) - Method in interface org.xml.sax.[ContentHandler](#)

Receive notification of character data.

[characters\(char\[\], int, int\)](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Receive notification of character data inside an element.

[characters\(char\[\], int, int\)](#) - Method in interface org.xml.sax.[DocumentHandler](#)

Deprecated. Receive notification of character data.

[characters\(char\[\], int, int\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Adapt a SAX2 characters event.

[characters\(char\[\], int, int\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter a character data event.

[characters\(char\[\], int, int\)](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of character data inside an element.

[characters\(char\[\], int, int\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Adapt a SAX1 characters event.

[clear\(\)](#) - Method in class org.xml.sax.helpers.[AttributeListImpl](#)

Deprecated. Clear the attribute list.

[clear\(\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Clear the attribute list for reuse.

[**clearParameters\(\)**](#) - Method in class `javax.xml.transform`.[Transformer](#)

Clear all parameters set with `setParameter`.

[**cloneNode\(boolean\)**](#) - Method in interface `org.w3c.dom`.[Node](#)

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes.

[**code**](#) - Variable in class `org.w3c.dom`.[DOMException](#)

[**Comment**](#) - interface `org.w3c.dom`.[Comment](#).

This interface inherits from `CharacterData` and represents the content of a comment, i.e., all the characters between the starting '`<!--`' and ending '`-->`'.

[**COMMENT_NODE**](#) - Static variable in interface `org.w3c.dom`.[Node](#)

The node is a `Comment`.

[**comment\(char\[\], int, int\)**](#) - Method in interface `org.xml.sax.ext`.[LexicalHandler](#)

Report an XML comment anywhere in the document.

[**ContentHandler**](#) - interface `org.xml.sax`.[ContentHandler](#).

Receive notification of the logical content of a document.

[**createAttribute\(String\)**](#) - Method in interface `org.w3c.dom`.[Document](#)

Creates an `Attr` of the given name.

[**createAttributeNS\(String, String\)**](#) - Method in interface `org.w3c.dom`.[Document](#)

Creates an attribute of the given qualified name and namespace URI.

[**createCDATASection\(String\)**](#) - Method in interface `org.w3c.dom`.[Document](#)

Creates a `CDATASection` node whose value is the specified string.

[**createComment\(String\)**](#) - Method in interface `org.w3c.dom`.[Document](#)

Creates a `Comment` node given the specified string.

[**createDocument\(String, String, DocumentType\)**](#) - Method in interface `org.w3c.dom`.[DOMImplementation](#)

Creates an XML `Document` object of the specified type with its document element.

[**createDocumentFragment\(\)**](#) - Method in interface `org.w3c.dom`.[Document](#)

Creates an empty `DocumentFragment` object.

[**createDocumentType\(String, String, String\)**](#) - Method in interface `org.w3c.dom`.[DOMImplementation](#)

Creates an empty `DocumentType` node.

[**createElement\(String\)**](#) - Method in interface `org.w3c.dom`.[Document](#)

Creates an element of the type specified.

[**createElementNS\(String, String\)**](#) - Method in interface `org.w3c.dom`.[Document](#)

Creates an element of the given qualified name and namespace URI.

[**createEntityReference\(String\)**](#) - Method in interface `org.w3c.dom`.[Document](#)

Creates an `EntityReference` object.

[createProcessingInstruction\(String, String\)](#) - Method in interface `org.w3c.dom.Document`

Creates a `ProcessingInstruction` node given the specified name and data strings.

[createTextNode\(String\)](#) - Method in interface `org.w3c.dom.Document`

Creates a `Text` node given the specified string.

[createXMLReader\(\)](#) - Static method in class `org.xml.sax.helpers.XMLReaderFactory`

Attempt to create an XML reader from a system property.

[createXMLReader\(String\)](#) - Static method in class `org.xml.sax.helpers.XMLReaderFactory`

Attempt to create an XML reader from a class name.

D

[declarePrefix\(String, String\)](#) - Method in class `org.xml.sax.helpers.NamespaceSupport`

Declare a Namespace prefix.

[DeclHandler](#) - interface `org.xml.sax.ext.DeclHandler`.

SAX2 extension handler for DTD declaration events.

[DefaultHandler](#) - class `org.xml.sax.helpers.DefaultHandler`.

Default base class for SAX2 event handlers.

[DefaultHandler\(\)](#) - Constructor for class `org.xml.sax.helpers.DefaultHandler`

[deleteData\(int, int\)](#) - Method in interface `org.w3c.dom.CharacterData`

Remove a range of 16-bit units from the node.

[DOCTYPE_PUBLIC](#) - Static variable in class `javax.xml.transform.OutputKeys`

doctype-public = string.

[DOCTYPE_SYSTEM](#) - Static variable in class `javax.xml.transform.OutputKeys`

doctype-system = string.

[Document](#) - interface `org.w3c.dom.Document`.

The `Document` interface represents the entire HTML or XML document.

[DOCUMENT_FRAGMENT_NODE](#) - Static variable in interface `org.w3c.dom.Node`

The node is a `DocumentFragment`.

[DOCUMENT_NODE](#) - Static variable in interface `org.w3c.dom.Node`

The node is a `Document`.

[DOCUMENT_TYPE_NODE](#) - Static variable in interface `org.w3c.dom.Node`

The node is a `DocumentType`.

[DocumentBuilder](#) - class javax.xml.parsers.[DocumentBuilder](#).

Defines the API to obtain DOM Document instances from an XML document.

[DocumentBuilder\(\)](#) - Constructor for class javax.xml.parsers.[DocumentBuilder](#)

[DocumentBuilderFactory](#) - class javax.xml.parsers.[DocumentBuilderFactory](#).

Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.

[DocumentBuilderFactory\(\)](#) - Constructor for class javax.xml.parsers.[DocumentBuilderFactory](#)

[DocumentFragment](#) - interface org.w3c.dom.[DocumentFragment](#).

DocumentFragment is a "lightweight" or "minimal" Document object.

[DocumentHandler](#) - interface org.xml.sax.[DocumentHandler](#).

Deprecated. *This interface has been replaced by the SAX2 [ContentHandler](#) interface, which includes Namespace support.*

[DocumentType](#) - interface org.w3c.dom.[DocumentType](#).

Each Document has a doctype attribute whose value is either null or a DocumentType object.

[DOMException](#) - exception org.w3c.dom.[DOMException](#).

DOM operations only raise exceptions in "exceptional" circumstances, i.e., when an operation is impossible to perform (either for logical reasons, because data is lost, or because the implementation has become unstable).

[DOMException\(short, String\)](#) - Constructor for class org.w3c.dom.[DOMException](#)

[DOMImplementation](#) - interface org.w3c.dom.[DOMImplementation](#).

The DOMImplementation interface provides a number of methods for performing operations that are independent of any particular instance of the document object model.

[DOMLocator](#) - interface javax.xml.transform.dom.[DOMLocator](#).

Indicates the position of a node in a source DOM, intended primarily for error reporting.

[DOMResult](#) - class javax.xml.transform.dom.[DOMResult](#).

Acts as a holder for a transformation result tree, in the form of a Document Object Model (DOM) tree.

[DOMResult\(\)](#) - Constructor for class javax.xml.transform.dom.[DOMResult](#)

Zero-argument default constructor.

[DOMResult\(Node\)](#) - Constructor for class javax.xml.transform.dom.[DOMResult](#)

Use a DOM node to create a new output target.

[DOMResult\(Node, String\)](#) - Constructor for class javax.xml.transform.dom.[DOMResult](#)

Create a new output target with a DOM node.

[**DOMSource**](#) - class javax.xml.transform.dom.[DOMSource](#).

Acts as a holder for a transformation Source tree in the form of a Document Object Model (DOM) tree.

[**DOMSource\(\)**](#) - Constructor for class javax.xml.transform.dom.[DOMSource](#)

Zero-argument default constructor.

[**DOMSource\(Node\)**](#) - Constructor for class javax.xml.transform.dom.[DOMSource](#)

Create a new input source with a DOM node.

[**DOMSource\(Node, String\)**](#) - Constructor for class javax.xml.transform.dom.[DOMSource](#)

Create a new input source with a DOM node, and with the system ID also passed in as the base URI.

[**DOMSTRING_SIZE_ERR**](#) - Static variable in class org.w3c.dom.[DOMException](#)

If the specified range of text does not fit into a DOMString

[**DTDHandler**](#) - interface org.xml.sax.[DTDHandler](#).

Receive notification of basic DTD-related events.

E

[**Element**](#) - interface org.w3c.dom.[Element](#).

The `Element` interface represents an element in an HTML or XML document.

[**ELEMENT_NODE**](#) - Static variable in interface org.w3c.dom.[Node](#)

The node is an `Element`.

[**elementDecl\(String, String\)**](#) - Method in interface org.xml.sax.ext.[DeclHandler](#)

Report an element type declaration.

[**ENCODING**](#) - Static variable in class javax.xml.transform.[OutputKeys](#)

encoding = string.

[**endCDATA\(\)**](#) - Method in interface org.xml.sax.ext.[LexicalHandler](#)

Report the end of a CDATA section.

[**endDocument\(\)**](#) - Method in interface org.xml.sax.[ContentHandler](#)

Receive notification of the end of a document.

[**endDocument\(\)**](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Receive notification of the end of the document.

[**endDocument\(\)**](#) - Method in interface org.xml.sax.[DocumentHandler](#)

Deprecated. Receive notification of the end of a document.

[**endDocument\(\)**](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

End document event.

[endDocument\(\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter an end document event.

[endDocument\(\)](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of the end of the document.

[endDocument\(\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Adapt a SAX1 end document event.

[endDTD\(\)](#) - Method in interface org.xml.sax.ext.[LexicalHandler](#)

Report the end of DTD declarations.

[endElement\(String\)](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Receive notification of the end of an element.

[endElement\(String\)](#) - Method in interface org.xml.sax.[DocumentHandler](#)

Deprecated. Receive notification of the end of an element.

[endElement\(String\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Adapt a SAX1 end element event.

[endElement\(String, String, String\)](#) - Method in interface org.xml.sax.[ContentHandler](#)

Receive notification of the end of an element.

[endElement\(String, String, String\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Adapt a SAX2 end element event.

[endElement\(String, String, String\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter an end element event.

[endElement\(String, String, String\)](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of the end of an element.

[endEntity\(String\)](#) - Method in interface org.xml.sax.ext.[LexicalHandler](#)

Report the end of an entity.

[endPrefixMapping\(String\)](#) - Method in interface org.xml.sax.[ContentHandler](#)

End the scope of a prefix-URI mapping.

[endPrefixMapping\(String\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Adapt a SAX2 end prefix mapping event.

[endPrefixMapping\(String\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter an end Namespace prefix mapping event.

[endPrefixMapping\(String\)](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of the end of a Namespace mapping.

[Entity](#) - interface org.w3c.dom.[Entity](#).

This interface represents an entity, either parsed or unparsed, in an XML document.

[ENTITY_NODE](#) - Static variable in interface org.w3c.dom.[Node](#)

The node is an Entity.

[ENTITY_REFERENCE_NODE](#) - Static variable in interface org.w3c.dom.[Node](#)

The node is an EntityReference.

[EntityReference](#) - interface org.w3c.dom.[EntityReference](#).

EntityReference objects may be inserted into the structure model when an entity reference is in the source document, or when the user wishes to insert an entity reference.

[EntityResolver](#) - interface org.xml.sax.[EntityResolver](#).

Basic interface for resolving entities.

[error\(SAXParseException\)](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Receive notification of a recoverable parser error.

[error\(SAXParseException\)](#) - Method in interface org.xml.sax.[ErrorHandler](#)

Receive notification of a recoverable error.

[error\(SAXParseException\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter an error event.

[error\(SAXParseException\)](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of a recoverable parser error.

[error\(TransformerException\)](#) - Method in interface javax.xml.transform.[ErrorListener](#)

Receive notification of a recoverable error.

[ErrorHandler](#) - interface org.xml.sax.[ErrorHandler](#).

Basic interface for SAX error handlers.

[ErrorListener](#) - interface javax.xml.transform.[ErrorListener](#).

To provide customized error handling, implement this interface and use the setErrorListener method to register an instance of the implementation with the [Transformer](#).

[externalEntityDecl\(String, String, String\)](#) - Method in interface org.xml.sax.ext.[DeclHandler](#)

Report a parsed external entity declaration.

F

[FactoryConfigurationError](#) - error javax.xml.parsers.[FactoryConfigurationError](#).

Thrown when a problem with configuration with the Parser Factories exists.

[FactoryConfigurationError\(\)](#) - Constructor for class javax.xml.parsers.[FactoryConfigurationError](#)

Create a new FactoryConfigurationError with no detail message.

[FactoryConfigurationError\(Exception\)](#) - Constructor for class

javax.xml.parsers.[FactoryConfigurationError](#)

Create a new `FactoryConfigurationError` with a given `Exception` base cause of the error.

[FactoryConfigurationError\(Exception, String\)](#) - Constructor for class

`javax.xml.parsers.FactoryConfigurationError`

Create a new `FactoryConfigurationError` with the given `Exception` base cause and detail message.

[FactoryConfigurationError\(String\)](#) - Constructor for class

`javax.xml.parsers.FactoryConfigurationError`

Create a new `FactoryConfigurationError` with the `String` specified as an error message.

[fatalError\(SAXParseException\)](#) - Method in class `org.xml.sax.HandlerBase`

Deprecated. Report a fatal XML parsing error.

[fatalError\(SAXParseException\)](#) - Method in interface `org.xml.sax.ErrorHandler`

Receive notification of a non-recoverable error.

[fatalError\(SAXParseException\)](#) - Method in class `org.xml.sax.helpers.XMLFilterImpl`

Filter a fatal error event.

[fatalError\(SAXParseException\)](#) - Method in class `org.xml.sax.helpers.DefaultHandler`

Report a fatal XML parsing error.

[fatalError\(TransformerException\)](#) - Method in interface `javax.xml.transform.ErrorListener`

Receive notification of a non-recoverable error.

[FEATURE](#) - Static variable in class `javax.xml.transform.dom.DOMSource`

If `TransformerFactory.getFeature\(java.lang.String\)` returns true when passed this value as an argument, the Transformer supports Source input of this type.

[FEATURE](#) - Static variable in class `javax.xml.transform.dom.DOMResult`

If `TransformerFactory.getFeature\(java.lang.String\)` returns true when passed this value as an argument, the Transformer supports Result output of this type.

[FEATURE](#) - Static variable in class `javax.xml.transform.sax.SAXResult`

If `TransformerFactory.getFeature\(java.lang.String\)` returns true when passed this value as an argument, the Transformer supports Result output of this type.

[FEATURE](#) - Static variable in class `javax.xml.transform.sax.SAXTransformerFactory`

If `TransformerFactory.getFeature\(java.lang.String\)` returns true when passed this value as an argument, the TransformerFactory returned from `TransformerFactory.newInstance\(\)` may be safely cast to a `SAXTransformerFactory`.

[FEATURE](#) - Static variable in class `javax.xml.transform.sax.SAXSource`

If `TransformerFactory.getFeature\(java.lang.String\)` returns true when passed this value as an argument, the Transformer supports Source input of this type.

[FEATURE](#) - Static variable in class `javax.xml.transform.stream.StreamResult`

If [TransformerFactory.getFeature\(java.lang.String\)](#) returns true when passed this value as an argument, the Transformer supports Result output of this type.

[FEATURE](#) - Static variable in class javax.xml.transform.stream.[StreamSource](#)

If [TransformerFactory.getFeature\(java.lang.String\)](#) returns true when passed this value as an argument, the Transformer supports Source input of this type.

[FEATURE_XMLFILTER](#) - Static variable in class javax.xml.transform.sax.[SAXTransformerFactory](#)

If [TransformerFactory.getFeature\(java.lang.String\)](#) returns true when passed this value as an argument, the [SAXTransformerFactory.newXMLFilter\(Source src\)](#) and [SAXTransformerFactory.newXMLFilter\(Templates templates\)](#) methods are supported.

G

[getAssociatedStylesheet\(Source, String, String, String\)](#) - Method in class javax.xml.transform.[TransformerFactory](#)

Get the stylesheet specification(s) associated via the xml-stylesheet processing instruction (see <http://www.w3.org/TR/xml-stylesheet/>) with the document document specified in the source parameter, and that match the given criteria.

[getAttribute\(String\)](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Allows the user to retrieve specific attributes on the underlying implementation.

[getAttribute\(String\)](#) - Method in class javax.xml.transform.[TransformerFactory](#)

Allows the user to retrieve specific attributes on the underlying implementation.

[getAttribute\(String\)](#) - Method in interface org.w3c.dom.[Element](#)

Retrieves an attribute value by name.

[getAttributeNode\(String\)](#) - Method in interface org.w3c.dom.[Element](#)

Retrieves an attribute node by name.

[getAttributeNodeNS\(String, String\)](#) - Method in interface org.w3c.dom.[Element](#)

Retrieves an `Attr` node by local name and namespace URI.

[getAttributeNS\(String, String\)](#) - Method in interface org.w3c.dom.[Element](#)

Retrieves an attribute value by local name and namespace URI.

[getAttributes\(\)](#) - Method in interface org.w3c.dom.[Node](#)

A `NamedNodeMap` containing the attributes of this node (if it is an `Element`) or null otherwise.

[getByteStream\(\)](#) - Method in class org.xml.sax.[InputSource](#)

Get the byte stream for this input source.

[getCause\(\)](#) - Method in class javax.xml.transform.[TransformerException](#)

Returns the cause of this throwable or `null` if the cause is nonexistent or unknown.

[**getCharacterStream\(\)**](#) - Method in class `org.xml.sax`.[InputSource](#)

Get the character stream for this input source.

[**getChildNodes\(\)**](#) - Method in interface `org.w3c.dom`.[Node](#)

A `NodeList` that contains all children of this node.

[**getColumnNumber\(\)**](#) - Method in interface `javax.xml.transform`.[SourceLocator](#)

Return the character position where the current document event ends.

[**getColumnNumber\(\)**](#) - Method in interface `org.xml.sax`.[Locator](#)

Return the column number where the current document event ends.

[**getColumnNumber\(\)**](#) - Method in class `org.xml.sax`.[SAXParseException](#)

The column number of the end of the text where the exception occurred.

[**getColumnNumber\(\)**](#) - Method in class `org.xml.sax.helpers`.[LocatorImpl](#)

Return the saved column number (1-based).

[**getContentHandler\(\)**](#) - Method in interface `org.xml.sax`.[XMLReader](#)

Return the current content handler.

[**getContentHandler\(\)**](#) - Method in class `org.xml.sax.helpers`.[XMLFilterImpl](#)

Get the content event handler.

[**getContentHandler\(\)**](#) - Method in class `org.xml.sax.helpers`.[ParserAdapter](#)

Return the current content handler.

[**getData\(\)**](#) - Method in interface `org.w3c.dom`.[ProcessingInstruction](#)

The content of this processing instruction.

[**getData\(\)**](#) - Method in interface `org.w3c.dom`.[CharacterData](#)

The character data of the node that implements this interface.

[**getDeclaredPrefixes\(\)**](#) - Method in class `org.xml.sax.helpers`.[NamespaceSupport](#)

Return an enumeration of all prefixes declared in this context.

[**getDoctype\(\)**](#) - Method in interface `org.w3c.dom`.[Document](#)

The Document Type Declaration (see `DocumentType`) associated with this document.

[**getDocumentElement\(\)**](#) - Method in interface `org.w3c.dom`.[Document](#)

This is a convenience attribute that allows direct access to the child node that is the root element of the document.

[**getDOMImplementation\(\)**](#) - Method in class `javax.xml.parsers`.[DocumentBuilder](#)

Obtain an instance of a [DOMImplementation](#) object.

[**getDTDHandler\(\)**](#) - Method in interface `org.xml.sax`.[XMLReader](#)

Return the current DTD handler.

[**getDTDHandler\(\)**](#) - Method in class `org.xml.sax.helpers`.[XMLFilterImpl](#)

Get the current DTD event handler.

[getDTDHandler\(\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Return the current DTD handler.

[getElementById\(String\)](#) - Method in interface org.w3c.dom.[Document](#)

Returns the `Element` whose ID is given by `elementId`.

[getElementsByTagName\(String\)](#) - Method in interface org.w3c.dom.[Element](#)

Returns a `NodeList` of all descendant `Elements` with a given tag name, in the order in which they are encountered in a preorder traversal of this `Element` tree.

[getElementsByTagName\(String\)](#) - Method in interface org.w3c.dom.[Document](#)

Returns a `NodeList` of all the `Elements` with a given tag name in the order in which they are encountered in a preorder traversal of the `Document` tree.

[getElementsByTagNameNS\(String, String\)](#) - Method in interface org.w3c.dom.[Element](#)

Returns a `NodeList` of all the descendant `Elements` with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of this `Element` tree.

[getElementsByTagNameNS\(String, String\)](#) - Method in interface org.w3c.dom.[Document](#)

Returns a `NodeList` of all the `Elements` with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of the `Document` tree.

[getEncoding\(\)](#) - Method in class org.xml.sax.[InputSource](#)

Get the character encoding for a byte stream or URI.

[getEntities\(\)](#) - Method in interface org.w3c.dom.[DocumentType](#)

A `NamedNodeMap` containing the general entities, both external and internal, declared in the DTD.

[getEntityResolver\(\)](#) - Method in interface org.xml.sax.[XMLReader](#)

Return the current entity resolver.

[getEntityResolver\(\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Get the current entity resolver.

[getEntityResolver\(\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Return the current entity resolver.

[getErrorHandler\(\)](#) - Method in interface org.xml.sax.[XMLReader](#)

Return the current error handler.

[getErrorHandler\(\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Get the current error event handler.

[getErrorHandler\(\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Return the current error handler.

[getErrorListener\(\)](#) - Method in class javax.xml.transform.[Transformer](#)

Get the error event handler in effect for the transformation.

[**getErrorListener\(\)**](#) - Method in class javax.xml.transform.[TransformerFactory](#)

Get the error event handler for the TransformerFactory.

[**getException\(\)**](#) - Method in class javax.xml.parsers.[FactoryConfigurationError](#)

Return the actual exception (if any) that caused this exception to be raised.

[**getException\(\)**](#) - Method in class javax.xml.transform.[TransformerException](#)

This method retrieves an exception that this exception wraps.

[**getException\(\)**](#) - Method in class javax.xml.transform.[TransformerFactoryConfigurationError](#)

Return the actual exception (if any) that caused this exception to be raised.

[**getException\(\)**](#) - Method in class org.xml.sax.[SAXException](#)

Return the embedded exception, if any.

[**getFeature\(String\)**](#) - Method in class javax.xml.parsers.[SAXParserFactory](#)

Returns the particular property requested for in the underlying implementation of org.xml.sax.XMLReader.

[**getFeature\(String\)**](#) - Method in class javax.xml.transform.[TransformerFactory](#)

Look up the value of a feature.

[**getFeature\(String\)**](#) - Method in interface org.xml.sax.[XMLReader](#)

Look up the value of a feature.

[**getFeature\(String\)**](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Look up the state of a feature.

[**getFeature\(String\)**](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Check a parser feature.

[**getFirstChild\(\)**](#) - Method in interface org.w3c.dom.[Node](#)

The first child of this node.

[**getHandler\(\)**](#) - Method in class javax.xml.transform.sax.[SAXResult](#)

Get the [ContentHandler](#) that is the Result.

[**getImplementation\(\)**](#) - Method in interface org.w3c.dom.[Document](#)

The DOMImplementation object that handles this document.

[**getIndex\(String\)**](#) - Method in interface org.xml.sax.[Attributes](#)

Look up the index of an attribute by XML 1.0 qualified name.

[**getIndex\(String\)**](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Look up an attribute's index by qualified (prefixed) name.

[**getIndex\(String, String\)**](#) - Method in interface org.xml.sax.[Attributes](#)

Look up the index of an attribute by Namespace name.

[**getIndex\(String, String\)**](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Look up an attribute's index by Namespace name.

[getInputSource\(\)](#) - Method in class javax.xml.transform.sax.[SAXSource](#)

Get the SAX InputSource to be used for the Source.

[getInputStream\(\)](#) - Method in class javax.xml.transform.stream.[StreamSource](#)

Get the byte stream that was set with setByteStream.

[getInternalSubset\(\)](#) - Method in interface org.w3c.dom.[DocumentType](#)

The internal subset as a string. The actual content returned depends on how much information is available to the implementation.

[getLastChild\(\)](#) - Method in interface org.w3c.dom.[Node](#)

The last child of this node.

[getLength\(\)](#) - Method in interface org.w3c.dom.[NamedNodeMap](#)

The number of nodes in this map.

[getLength\(\)](#) - Method in interface org.w3c.dom.[CharacterData](#)

The number of 16-bit units that are available through data and the substringData method below.

[getLength\(\)](#) - Method in interface org.w3c.dom.[NodeList](#)

The number of nodes in the list.

[getLength\(\)](#) - Method in interface org.xml.sax.[Attributes](#)

Return the number of attributes in the list.

[getLength\(\)](#) - Method in interface org.xml.sax.[AttributeList](#)

Deprecated. Return the number of attributes in this list.

[getLength\(\)](#) - Method in class org.xml.sax.helpers.[AttributeListImpl](#)

Deprecated. Return the number of attributes in the list.

[getLength\(\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Return the number of attributes in the list.

[getLexicalHandler\(\)](#) - Method in class javax.xml.transform.sax.[SAXResult](#)

Get a SAX2 [LexicalHandler](#) for the output.

[getLineNumber\(\)](#) - Method in interface javax.xml.transform.[SourceLocator](#)

Return the line number where the current document event ends.

[getLineNumber\(\)](#) - Method in interface org.xml.sax.[Locator](#)

Return the line number where the current document event ends.

[getLineNumber\(\)](#) - Method in class org.xml.sax.[SAXParseException](#)

The line number of the end of the text where the exception occurred.

[getLineNumber\(\)](#) - Method in class org.xml.sax.helpers.[LocatorImpl](#)

Return the saved line number (1-based).

[getLocalName\(\)](#) - Method in interface org.w3c.dom.[Node](#)

Returns the local part of the qualified name of this node.

[getLocalName\(int\)](#) - Method in interface org.xml.sax.[Attributes](#)

Look up an attribute's local name by index.

[getLocalName\(int\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Return an attribute's local name.

[getLocationAsString\(\)](#) - Method in class javax.xml.transform.[TransformerException](#)

Get the location information as a string.

[getLocator\(\)](#) - Method in class javax.xml.transform.[TransformerException](#)

Method `getLocator` retrieves an instance of a `SourceLocator` object that specifies where an error occurred.

[getMessage\(\)](#) - Method in class javax.xml.parsers.[FactoryConfigurationError](#)

Return the message (if any) for this error .

[getMessage\(\)](#) - Method in class javax.xml.transform.[TransformerFactoryConfigurationError](#)

Return the message (if any) for this error .

[getMessage\(\)](#) - Method in class org.xml.sax.[SAXException](#)

Return a detail message for this exception.

[getMessageAndLocation\(\)](#) - Method in class javax.xml.transform.[TransformerException](#)

Get the error message with location information appended.

[getName\(\)](#) - Method in interface org.w3c.dom.[Attr](#)

Returns the name of this attribute.

[getName\(\)](#) - Method in interface org.w3c.dom.[DocumentType](#)

The name of DTD; i.e., the name immediately following the DOCTYPE keyword.

[getName\(int\)](#) - Method in interface org.xml.sax.[AttributeList](#)

Deprecated. Return the name of an attribute in this list (by position).

[getName\(int\)](#) - Method in class org.xml.sax.helpers.[AttributeListImpl](#)

Deprecated. Get the name of an attribute (by position).

[getNamedItem\(String\)](#) - Method in interface org.w3c.dom.[NamedNodeMap](#)

Retrieves a node specified by name.

[getNamedItemNS\(String, String\)](#) - Method in interface org.w3c.dom.[NamedNodeMap](#)

Retrieves a node specified by local name and namespace URI.

[getNamespaceURI\(\)](#) - Method in interface org.w3c.dom.[Node](#)

The namespace URI of this node, or `null` if it is unspecified.

[getNextSibling\(\)](#) - Method in interface org.w3c.dom.[Node](#)

The node immediately following this node.

[getNode\(\)](#) - Method in class javax.xml.transform.dom.[DOMSource](#)

Get the node that represents a Source DOM tree.

[**getNode\(\)**](#) - Method in class javax.xml.transform.dom.[DOMResult](#)

Get the node that will contain the result DOM tree.

[**getNodeName\(\)**](#) - Method in interface org.w3c.dom.[Node](#)

The name of this node, depending on its type; see the table above.

[**getNodeType\(\)**](#) - Method in interface org.w3c.dom.[Node](#)

A code representing the type of the underlying object, as defined above.

[**getNodeValue\(\)**](#) - Method in interface org.w3c.dom.[Node](#)

The value of this node, depending on its type; see the table above.

[**getNotationName\(\)**](#) - Method in interface org.w3c.dom.[Entity](#)

For unparsed entities, the name of the notation for the entity.

[**getNotations\(\)**](#) - Method in interface org.w3c.dom.[DocumentType](#)

A NamedNodeMap containing the notations declared in the DTD.

[**getOriginatingNode\(\)**](#) - Method in interface javax.xml.transform.dom.[DOMLocator](#)

Return the node where the event occurred.

[**getOutputProperties\(\)**](#) - Method in class javax.xml.transform.[Transformer](#)

Get a copy of the output properties for the transformation.

[**getOutputProperties\(\)**](#) - Method in interface javax.xml.transform.[Templates](#)

Get the static properties for xsl:output.

[**getOutputProperty\(String\)**](#) - Method in class javax.xml.transform.[Transformer](#)

Get an output property that is in effect for the transformation.

[**getOutputStream\(\)**](#) - Method in class javax.xml.transform.stream.[StreamResult](#)

Get the byte stream that was set with setOutputStream.

[**getOwnerDocument\(\)**](#) - Method in interface org.w3c.dom.[Node](#)

The Document object associated with this node.

[**getOwnerElement\(\)**](#) - Method in interface org.w3c.dom.[Attr](#)

The Element node this attribute is attached to or null if this attribute is not in use.

[**getParameter\(String\)**](#) - Method in class javax.xml.transform.[Transformer](#)

Get a parameter that was explicitly set with setParameter or setParameters.

[**getParent\(\)**](#) - Method in interface org.xml.sax.[XMLFilter](#)

Get the parent reader.

[**getParent\(\)**](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Get the parent reader.

[**getParentNode\(\)**](#) - Method in interface org.w3c.dom.[Node](#)

The parent of this node.

[getParser\(\)](#) - Method in class javax.xml.parsers.[SAXParser](#)

Returns the SAX parser that is encapsulated by the implementation of this class.

[getPrefix\(\)](#) - Method in interface org.w3c.dom.[Node](#)

The namespace prefix of this node, or null if it is unspecified.

[getPrefix\(String\)](#) - Method in class org.xml.sax.helpers.[NamespaceSupport](#)

Return one of the prefixes mapped to a Namespace URI.

[getPrefixes\(\)](#) - Method in class org.xml.sax.helpers.[NamespaceSupport](#)

Return an enumeration of all prefixes currently declared.

[getPrefixes\(String\)](#) - Method in class org.xml.sax.helpers.[NamespaceSupport](#)

Return an enumeration of all prefixes currently declared for a URI.

[getPreviousSibling\(\)](#) - Method in interface org.w3c.dom.[Node](#)

The node immediately preceding this node.

[getProperty\(String\)](#) - Method in class javax.xml.parsers.[SAXParser](#)

Returns the particular property requested for in the underlying implementation of [XMLReader](#).

[getProperty\(String\)](#) - Method in interface org.xml.sax.[XMLReader](#)

Look up the value of a property.

[getProperty\(String\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Look up the value of a property.

[getProperty\(String\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Get a parser property.

[getPublicId\(\)](#) - Method in interface javax.xml.transform.[SourceLocator](#)

Return the public identifier for the current document event.

[getPublicId\(\)](#) - Method in class javax.xml.transform.stream.[StreamSource](#)

Get the public identifier that was set with setPublicId.

[getPublicId\(\)](#) - Method in interface org.w3c.dom.[Notation](#)

The public identifier of this notation.

[getPublicId\(\)](#) - Method in interface org.w3c.dom.[Entity](#)

The public identifier associated with the entity, if specified.

[getPublicId\(\)](#) - Method in interface org.w3c.dom.[DocumentType](#)

The public identifier of the external subset.

[getPublicId\(\)](#) - Method in interface org.xml.sax.[Locator](#)

Return the public identifier for the current document event.

[getPublicId\(\)](#) - Method in class org.xml.sax.[SAXParseException](#)

Get the public identifier of the entity where the exception occurred.

[getPublicId\(\)](#) - Method in class org.xml.sax.[InputSource](#)

Get the public identifier for this input source.

[getPublicId\(\)](#) - Method in class org.xml.sax.helpers.[LocatorImpl](#)

Return the saved public identifier.

[getQName\(int\)](#) - Method in interface org.xml.sax.[Attributes](#)

Look up an attribute's XML 1.0 qualified name by index.

[getQName\(int\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Return an attribute's qualified (prefixed) name.

[getReader\(\)](#) - Method in class javax.xml.transform.stream.[StreamSource](#)

Get the character stream that was set with setReader.

[getSpecified\(\)](#) - Method in interface org.w3c.dom.[Attr](#)

If this attribute was explicitly given a value in the original document, this is `true`; otherwise, it is `false`.

[getSystemId\(\)](#) - Method in interface javax.xml.transform.[Result](#)

Get the system identifier that was set with setSystemId.

[getSystemId\(\)](#) - Method in interface javax.xml.transform.[SourceLocator](#)

Return the system identifier for the current document event.

[getSystemId\(\)](#) - Method in interface javax.xml.transform.[Source](#)

Get the system identifier that was set with setSystemId.

[getSystemId\(\)](#) - Method in class javax.xml.transform.dom.[DOMSource](#)

Get the base ID (URL or system ID) from where URLs will be resolved.

[getSystemId\(\)](#) - Method in class javax.xml.transform.dom.[DOMResult](#)

Get the system identifier that was set with setSystemId.

[getSystemId\(\)](#) - Method in interface javax.xml.transform.sax.[TransformerHandler](#)

Get the base ID (URI or system ID) from where relative URLs will be resolved.

[getSystemId\(\)](#) - Method in interface javax.xml.transform.sax.[TemplatesHandler](#)

Get the base ID (URI or system ID) from where relative URLs will be resolved.

[getSystemId\(\)](#) - Method in class javax.xml.transform.sax.[SAXResult](#)

Get the system identifier that was set with setSystemId.

[getSystemId\(\)](#) - Method in class javax.xml.transform.sax.[SAXSource](#)

Get the base ID (URI or system ID) from where URIs will be resolved.

[getSystemId\(\)](#) - Method in class javax.xml.transform.stream.[StreamResult](#)

Get the system identifier that was set with setSystemId.

[getSystemId\(\)](#) - Method in class javax.xml.transform.stream.[StreamSource](#)

Get the system identifier that was set with setSystemId.

[getSystemId\(\)](#) - Method in interface org.w3c.dom.[Notation](#)

The system identifier of this notation.

[getId\(\)](#) - Method in interface org.w3c.dom.[Entity](#)

The system identifier associated with the entity, if specified.

[getId\(\)](#) - Method in interface org.w3c.dom.[DocumentType](#)

The system identifier of the external subset.

[getId\(\)](#) - Method in interface org.xml.sax.[Locator](#)

Return the system identifier for the current document event.

[getId\(\)](#) - Method in class org.xml.sax.[SAXParseException](#)

Get the system identifier of the entity where the exception occurred.

[getId\(\)](#) - Method in class org.xml.sax.[InputSource](#)

Get the system identifier for this input source.

[getId\(\)](#) - Method in class org.xml.sax.helpers.[LocatorImpl](#)

Return the saved system identifier.

[getTagName\(\)](#) - Method in interface org.w3c.dom.[Element](#)

The name of the element.

[getTarget\(\)](#) - Method in interface org.w3c.dom.[ProcessingInstruction](#)

The target of this processing instruction.

[getTemplates\(\)](#) - Method in interface javax.xml.transform.sax.[TemplatesHandler](#)

When a TemplatesHandler object is used as a ContentHandler for the parsing of transformation instructions, it creates a Templates object, which the caller can get once the SAX events have been completed.

[getTransformer\(\)](#) - Method in interface javax.xml.transform.sax.[TransformerHandler](#)

Get the Transformer associated with this handler, which is needed in order to set parameters and output properties.

[getType\(int\)](#) - Method in interface org.xml.sax.[Attributes](#)

Look up an attribute's type by index.

[getType\(int\)](#) - Method in interface org.xml.sax.[AttributeList](#)

Deprecated. Return the type of an attribute in the list (by position).

[getType\(int\)](#) - Method in class org.xml.sax.helpers.[AttributeListImpl](#)

Deprecated. Get the type of an attribute (by position).

[getType\(int\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Return an attribute's type by index.

[getType\(String\)](#) - Method in interface org.xml.sax.[Attributes](#)

Look up an attribute's type by XML 1.0 qualified name.

[getType\(String\)](#) - Method in interface org.xml.sax.[AttributeList](#)

Deprecated. Return the type of an attribute in the list (by name).

[getType\(String\)](#) - Method in class org.xml.sax.helpers.[AttributeListImpl](#)

Deprecated. Get the type of an attribute (by name).

[getType\(String\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Look up an attribute's type by qualified (prefixed) name.

[getType\(String, String\)](#) - Method in interface org.xml.sax.[Attributes](#)

Look up an attribute's type by Namespace name.

[getType\(String, String\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Look up an attribute's type by Namespace-qualified name.

[getURI\(int\)](#) - Method in interface org.xml.sax.[Attributes](#)

Look up an attribute's Namespace URI by index.

[getURI\(int\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Return an attribute's Namespace URI.

[getURI\(String\)](#) - Method in class org.xml.sax.helpers.[NamespaceSupport](#)

Look up a prefix and get the currently-mapped Namespace URI.

[getURIResolver\(\)](#) - Method in class javax.xml.transform.[Transformer](#)

Get an object that will be used to resolve URIs used in document(), etc.

[getURIResolver\(\)](#) - Method in class javax.xml.transform.[TransformerFactory](#)

Get the object that is used by default during the transformation to resolve URIs used in document(), xsl:import, or xsl:include.

[getValue\(\)](#) - Method in interface org.w3c.dom.[Attr](#)

On retrieval, the value of the attribute is returned as a string.

[getValue\(int\)](#) - Method in interface org.xml.sax.[Attributes](#)

Look up an attribute's value by index.

[getValue\(int\)](#) - Method in interface org.xml.sax.[AttributeList](#)

Deprecated. Return the value of an attribute in the list (by position).

[getValue\(int\)](#) - Method in class org.xml.sax.helpers.[AttributeListImpl](#)

Deprecated. Get the value of an attribute (by position).

[getValue\(int\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Return an attribute's value by index.

[getValue\(String\)](#) - Method in interface org.xml.sax.[Attributes](#)

Look up an attribute's value by XML 1.0 qualified name.

[getValue\(String\)](#) - Method in interface org.xml.sax.[AttributeList](#)

Deprecated. Return the value of an attribute in the list (by name).

[getValue\(String\)](#) - Method in class org.xml.sax.helpers.[AttributeListImpl](#)

Deprecated. Get the value of an attribute (by name).

[**getValue\(String\)**](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Look up an attribute's value by qualified (prefixed) name.

[**getValue\(String, String\)**](#) - Method in interface org.xml.sax.[Attributes](#)

Look up an attribute's value by Namespace name.

[**getValue\(String, String\)**](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Look up an attribute's value by Namespace-qualified name.

[**getWriter\(\)**](#) - Method in class javax.xml.transform.stream.[StreamResult](#)

Get the character stream that was set with setWriter.

[**getXMLReader\(\)**](#) - Method in class javax.xml.parsers.[SAXParser](#)

Returns the [XMLReader](#) that is encapsulated by the implementation of this class.

[**getXMLReader\(\)**](#) - Method in class javax.xml.transform.sax.[SAXSource](#)

Get the XMLReader to be used for the Source.

H

[**HandlerBase**](#) - class org.xml.sax.[HandlerBase](#).

Deprecated. *This class works with the deprecated [DocumentHandler](#) interface. It has been replaced by the SAX2 [DefaultHandler](#) class.*

[**HandlerBase\(\)**](#) - Constructor for class org.xml.sax.[HandlerBase](#)

Deprecated.

[**hasAttribute\(String\)**](#) - Method in interface org.w3c.dom.[Element](#)

Returns `true` when an attribute with a given name is specified on this element or has a default value, `false` otherwise.

[**hasAttributeNS\(String, String\)**](#) - Method in interface org.w3c.dom.[Element](#)

Returns `true` when an attribute with a given local name and namespace URI is specified on this element or has a default value, `false` otherwise.

[**hasAttributes\(\)**](#) - Method in interface org.w3c.dom.[Node](#)

Returns whether this node (if it is an element) has any attributes.

[**hasChildNodes\(\)**](#) - Method in interface org.w3c.dom.[Node](#)

Returns whether this node has any children.

[**hasFeature\(String, String\)**](#) - Method in interface org.w3c.dom.[DOMImplementation](#)

Test if the DOM implementation implements a specific feature.

[**HIERARCHY_REQUEST_ERR**](#) - Static variable in class org.w3c.dom.[DOMException](#)

If any node is inserted somewhere it doesn't belong

[ignorableWhitespace\(char\[\], int, int\)](#) - Method in interface org.xml.sax.[ContentHandler](#)

Receive notification of ignorable whitespace in element content.

[ignorableWhitespace\(char\[\], int, int\)](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Receive notification of ignorable whitespace in element content.

[ignorableWhitespace\(char\[\], int, int\)](#) - Method in interface org.xml.sax.[DocumentHandler](#)

Deprecated. Receive notification of ignorable whitespace in element content.

[ignorableWhitespace\(char\[\], int, int\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Adapt a SAX2 ignorable whitespace event.

[ignorableWhitespace\(char\[\], int, int\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter an ignorable whitespace event.

[ignorableWhitespace\(char\[\], int, int\)](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of ignorable whitespace in element content.

[ignorableWhitespace\(char\[\], int, int\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Adapt a SAX1 ignorable whitespace event.

[importNode\(Node, boolean\)](#) - Method in interface org.w3c.dom.[Document](#)

Imports a node from another document to this document.

[INDENT](#) - Static variable in class javax.xml.transform.[OutputKeys](#)

indent = "yes" | "no".

[INDEX_SIZE_ERR](#) - Static variable in class org.w3c.dom.[DOMException](#)

If index or size is negative, or greater than the allowed value

[initCause\(Throwable\)](#) - Method in class javax.xml.transform.[TransformerException](#)

Initializes the *cause* of this throwable to the specified value.

[InputSource](#) - class org.xml.sax.[InputSource](#).

A single input source for an XML entity.

[InputSource\(\)](#) - Constructor for class org.xml.sax.[InputSource](#)

Zero-argument default constructor.

[InputSource\(InputStream\)](#) - Constructor for class org.xml.sax.[InputSource](#)

Create a new input source with a byte stream.

[InputSource\(Reader\)](#) - Constructor for class org.xml.sax.[InputSource](#)

Create a new input source with a character stream.

[InputSource\(String\)](#) - Constructor for class org.xml.sax.[InputSource](#)

Create a new input source with a system identifier.

[**insertBefore\(Node, Node\)**](#) - Method in interface org.w3c.dom.[Node](#)

Inserts the node `newChild` before the existing child node `refChild`.

[**insertData\(int, String\)**](#) - Method in interface org.w3c.dom.[CharacterData](#)

Insert a string at the specified 16-bit unit offset.

[**internalEntityDecl\(String, String\)**](#) - Method in interface org.xml.sax.ext.[DeclHandler](#)

Report an internal entity declaration.

[**INUSE_ATTRIBUTE_ERR**](#) - Static variable in class org.w3c.dom.[DOMException](#)

If an attempt is made to add an attribute that is already in use elsewhere

[**INVALID_ACCESS_ERR**](#) - Static variable in class org.w3c.dom.[DOMException](#)

If a parameter or an operation is not supported by the underlying object.

[**INVALID_CHARACTER_ERR**](#) - Static variable in class org.w3c.dom.[DOMException](#)

If an invalid or illegal character is specified, such as in a name.

[**INVALID_MODIFICATION_ERR**](#) - Static variable in class org.w3c.dom.[DOMException](#)

If an attempt is made to modify the type of the underlying object.

[**INVALID_STATE_ERR**](#) - Static variable in class org.w3c.dom.[DOMException](#)

If an attempt is made to use an object that is not, or is no longer, usable.

[**isCoalescing\(\)**](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Indicates whether or not the factory is configured to produce parsers which converts CDATA nodes to Text nodes and appends it to the adjacent (if any) Text node.

[**isExpandEntityReferences\(\)**](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Indicates whether or not the factory is configured to produce parsers which expand entity reference nodes.

[**isIgnoringComments\(\)**](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Indicates whether or not the factory is configured to produce parsers which ignores comments.

[**isIgnoringElementContentWhitespace\(\)**](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Indicates whether or not the factory is configured to produce parsers which ignore ignorable whitespace in element content.

[**isNamespaceAware\(\)**](#) - Method in class javax.xml.parsers.[SAXParser](#)

Indicates whether or not this parser is configured to understand namespaces.

[**isNamespaceAware\(\)**](#) - Method in class javax.xml.parsers.[DocumentBuilder](#)

Indicates whether or not this parser is configured to understand namespaces.

[**isNamespaceAware\(\)**](#) - Method in class javax.xml.parsers.[SAXParserFactory](#)

Indicates whether or not the factory is configured to produce parsers which are namespace aware.

[**isNamespaceAware\(\)**](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Indicates whether or not the factory is configured to produce parsers which are namespace aware.

[**isSupported\(String, String\)**](#) - Method in interface org.w3c.dom.[Node](#)

Tests whether the DOM implementation implements a specific feature and that feature is supported by this node.

[**isValidating\(\)**](#) - Method in class javax.xml.parsers.[SAXParser](#)

Indicates whether or not this parser is configured to validate XML documents.

[**isValidating\(\)**](#) - Method in class javax.xml.parsers.[DocumentBuilder](#)

Indicates whether or not this parser is configured to validate XML documents.

[**isValidating\(\)**](#) - Method in class javax.xml.parsers.[SAXParserFactory](#)

Indicates whether or not the factory is configured to produce parsers which validate the XML content during parse.

[**isValidating\(\)**](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Indicates whether or not the factory is configured to produce parsers which validate the XML content during parse.

[**item\(int\)**](#) - Method in interface org.w3c.dom.[NamedNodeMap](#)

Returns the `index`th item in the map.

[**item\(int\)**](#) - Method in interface org.w3c.dom.[NodeList](#)

Returns the `index`th item in the collection.

J

[javax.xml.parsers](#) - package javax.xml.parsers

Provides classes allowing the processing of XML documents.

[javax.xml.transform](#) - package javax.xml.transform

This package defines the generic APIs for processing transformation instructions, and performing a transformation from source to result.

[javax.xml.transform.dom](#) - package javax.xml.transform.dom

This package implements DOM-specific transformation APIs. The [DOMSource](#) class allows the client of the implementation of this API to specify a DOM [Node](#) as the source of the input tree.

[javax.xml.transform.sax](#) - package javax.xml.transform.sax

This package implements SAX2-specific transformation APIs.

[javax.xml.transform.stream](#) - package javax.xml.transform.stream

This package implements stream- and URI- specific transformation APIs.

L

[LexicalHandler](#) - interface org.xml.sax.ext.[LexicalHandler](#).

SAX2 extension handler for lexical events.

[Locator](#) - interface org.xml.sax.[Locator](#).

Interface for associating a SAX event with a document location.

[LocatorImpl](#) - class org.xml.sax.helpers.[LocatorImpl](#).

Provide an optional convenience implementation of Locator.

[LocatorImpl\(\)](#) - Constructor for class org.xml.sax.helpers.[LocatorImpl](#)

Zero-argument constructor.

[LocatorImpl\(Locator\)](#) - Constructor for class org.xml.sax.helpers.[LocatorImpl](#)

Copy constructor.

M

[makeParser\(\)](#) - Static method in class org.xml.sax.helpers.[ParserFactory](#)

Deprecated. Create a new SAX parser using the `org.xml.sax.parser' system property.

[makeParser\(String\)](#) - Static method in class org.xml.sax.helpers.[ParserFactory](#)

Deprecated. Create a new SAX parser object using the class name provided.

[MEDIA_TYPE](#) - Static variable in class javax.xml.transform.[OutputKeys](#)

media-type = string.

[METHOD](#) - Static variable in class javax.xml.transform.[OutputKeys](#)

method = "xml" | "html" | "text" | expanded name.

N

[NamedNodeMap](#) - interface org.w3c.dom.[NamedNodeMap](#).

Objects implementing the NamedNodeMap interface are used to represent collections of nodes that can be accessed by name.

[NAMESPACE_ERR](#) - Static variable in class org.w3c.dom.[DOMException](#)

If an attempt is made to create or change an object in a way which is incorrect with regard to namespaces.

[NamespaceSupport](#) - class org.xml.sax.helpers.[NamespaceSupport](#).

Encapsulate Namespace logic for use by SAX drivers.

[NamespaceSupport\(\)](#) - Constructor for class org.xml.sax.helpers.[NamespaceSupport](#)

Create a new Namespace support object.

[newDocument\(\)](#) - Method in class javax.xml.parsers.[DocumentBuilder](#)

Obtain a new instance of a DOM [Document](#) object to build a DOM tree with.

[newDocumentBuilder\(\)](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Creates a new instance of a [DocumentBuilder](#) using the currently configured parameters.

[newInstance\(\)](#) - Static method in class javax.xml.parsers.[SAXParserFactory](#)

Obtain a new instance of a SAXParserFactory.

[newInstance\(\)](#) - Static method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Obtain a new instance of a DocumentBuilderFactory.

[newInstance\(\)](#) - Static method in class javax.xml.transform.[TransformerFactory](#)

Obtain a new instance of a TransformerFactory.

[newSAXParser\(\)](#) - Method in class javax.xml.parsers.[SAXParserFactory](#)

Creates a new instance of a SAXParser using the currently configured factory parameters.

[newTemplates\(Source\)](#) - Method in class javax.xml.transform.[TransformerFactory](#)

Process the Source into a Templates object, which is a compiled representation of the source.

[newTemplatesHandler\(\)](#) - Method in class javax.xml.transform.sax.[SAXTransformerFactory](#)

Get a TemplatesHandler object that can process SAX ContentHandler events into a Templates object.

[newTransformer\(\)](#) - Method in interface javax.xml.transform.[Templates](#)

Create a new transformation context for this Templates object.

[newTransformer\(\)](#) - Method in class javax.xml.transform.[TransformerFactory](#)

Create a new Transformer object that performs a copy of the source to the result.

[newTransformer\(Source\)](#) - Method in class javax.xml.transform.[TransformerFactory](#)

Process the Source into a Transformer object.

[newTransformerHandler\(\)](#) - Method in class javax.xml.transform.sax.[SAXTransformerFactory](#)

Get a TransformerHandler object that can process SAX ContentHandler events into a Result.

[newTransformerHandler\(Source\)](#) - Method in class javax.xml.transform.sax.[SAXTransformerFactory](#)

Get a TransformerHandler object that can process SAX ContentHandler events into a Result, based on the transformation instructions specified by the argument.

[newTransformerHandler\(Templates\)](#) - Method in class

javax.xml.transform.sax.[SAXTransformerFactory](#)

Get a TransformerHandler object that can process SAX ContentHandler events into a Result, based on the Templates argument.

[newXMLFilter\(Source\)](#) - Method in class javax.xml.transform.sax.[SAXTransformerFactory](#)

Create an XMLFilter that uses the given Source as the transformation instructions.

[**newXMLFilter\(Templates\)**](#) - Method in class javax.xml.transform.sax.[SAXTransformerFactory](#)

Create an XMLFilter, based on the Templates argument..

[**NO_DATA_ALLOWED_ERR**](#) - Static variable in class org.w3c.dom.[DOMException](#)

If data is specified for a node which does not support data

[**NO_MODIFICATION_ALLOWED_ERR**](#) - Static variable in class org.w3c.dom.[DOMException](#)

If an attempt is made to modify an object where modifications are not allowed

[**Node**](#) - interface org.w3c.dom.[Node](#).

The Node interface is the primary datatype for the entire Document Object Model.

[**NodeList**](#) - interface org.w3c.dom.[NodeList](#).

The NodeList interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented.

[**normalize\(\)**](#) - Method in interface org.w3c.dom.[Node](#)

Puts all Text nodes in the full depth of the sub-tree underneath this Node, including attribute nodes, into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates Text nodes, i.e., there are neither adjacent Text nodes nor empty Text nodes.

[**NOT_FOUND_ERR**](#) - Static variable in class org.w3c.dom.[DOMException](#)

If an attempt is made to reference a node in a context where it does not exist

[**NOT_SUPPORTED_ERR**](#) - Static variable in class org.w3c.dom.[DOMException](#)

If the implementation does not support the requested type of object or operation.

[**Notation**](#) - interface org.w3c.dom.[Notation](#).

This interface represents a notation declared in the DTD.

[**NOTATION_NODE**](#) - Static variable in interface org.w3c.dom.[Node](#)

The node is a Notation.

[**notationDecl\(String, String, String\)**](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Receive notification of a notation declaration.

[**notationDecl\(String, String, String\)**](#) - Method in interface org.xml.sax.[DTDHandler](#)

Receive notification of a notation declaration event.

[**notationDecl\(String, String, String\)**](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter a notation declaration event.

[**notationDecl\(String, String, String\)**](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of a notation declaration.

O

[OMIT_XML_DECLARATION](#) - Static variable in class javax.xml.transform.[OutputKeys](#)

omit-xml-declaration = "yes" | "no".

[org.w3c.dom](#) - package org.w3c.dom

[org.xml.sax](#) - package org.xml.sax

[org.xml.sax.ext](#) - package org.xml.sax.ext

[org.xml.sax.helpers](#) - package org.xml.sax.helpers

[OutputKeys](#) - class javax.xml.transform.[OutputKeys](#).

Provides string constants that can be used to set output properties for a Transformer, or to retrieve output properties from a Transformer or Templates object.

P

[parse\(File\)](#) - Method in class javax.xml.parsers.[DocumentBuilder](#)

Parse the content of the given file as an XML document and return a new DOM [Document](#) object.

[parse\(File, DefaultHandler\)](#) - Method in class javax.xml.parsers.[SAXParser](#)

Parse the content of the file specified as XML using the specified [DefaultHandler](#).

[parse\(File, HandlerBase\)](#) - Method in class javax.xml.parsers.[SAXParser](#)

Parse the content of the file specified as XML using the specified [HandlerBase](#).

[parse\(InputSource\)](#) - Method in class javax.xml.parsers.[DocumentBuilder](#)

Parse the content of the given input source as an XML document and return a new DOM [Document](#) object.

[parse\(InputSource\)](#) - Method in interface org.xml.sax.[Parser](#)

Deprecated. Parse an XML document.

[parse\(InputSource\)](#) - Method in interface org.xml.sax.[XMLReader](#)

Parse an XML document.

[parse\(InputSource\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Parse the document.

[parse\(InputSource\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Parse a document.

[**parse\(InputSource\)**](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Parse an XML document.

[**parse\(InputSource, DefaultHandler\)**](#) - Method in class javax.xml.parsers.[SAXParser](#)

Parse the content given [InputSource](#) as XML using the specified [DefaultHandler](#).

[**parse\(InputSource, HandlerBase\)**](#) - Method in class javax.xml.parsers.[SAXParser](#)

Parse the content given [InputSource](#) as XML using the specified [HandlerBase](#).

[**parse\(InputStream\)**](#) - Method in class javax.xml.parsers.[DocumentBuilder](#)

Parse the content of the given `InputStream` as an XML document and return a new DOM [Document](#) object.

[**parse\(InputStream, DefaultHandler\)**](#) - Method in class javax.xml.parsers.[SAXParser](#)

Parse the content of the given `InputStream` instance as XML using the specified [DefaultHandler](#).

[**parse\(InputStream, DefaultHandler, String\)**](#) - Method in class javax.xml.parsers.[SAXParser](#)

Parse the content of the given `InputStream` instance as XML using the specified [DefaultHandler](#).

[**parse\(InputStream, HandlerBase\)**](#) - Method in class javax.xml.parsers.[SAXParser](#)

Parse the content of the given `InputStream` instance as XML using the specified [HandlerBase](#).

[**parse\(InputStream, HandlerBase, String\)**](#) - Method in class javax.xml.parsers.[SAXParser](#)

Parse the content of the given `InputStream` instance as XML using the specified [HandlerBase](#).

[**parse\(InputStream, String\)**](#) - Method in class javax.xml.parsers.[DocumentBuilder](#)

Parse the content of the given `InputStream` as an XML document and return a new DOM [Document](#) object.

[**parse\(String\)**](#) - Method in class javax.xml.parsers.[DocumentBuilder](#)

Parse the content of the given URI as an XML document and return a new DOM [Document](#) object.

[**parse\(String\)**](#) - Method in interface org.xml.sax.[Parser](#)

Deprecated. Parse an XML document from a system identifier (URI).

[**parse\(String\)**](#) - Method in interface org.xml.sax.[XMLReader](#)

Parse an XML document from a system identifier (URI).

[**parse\(String\)**](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Parse the document.

[**parse\(String\)**](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Parse a document.

[parse\(String\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Parse an XML document.

[parse\(String, DefaultHandler\)](#) - Method in class javax.xml.parsers.[SAXParser](#)

Parse the content described by the giving Uniform Resource Identifier (URI) as XML using the specified [DefaultHandler](#).

[parse\(String, HandlerBase\)](#) - Method in class javax.xml.parsers.[SAXParser](#)

Parse the content described by the giving Uniform Resource Identifier (URI) as XML using the specified [HandlerBase](#).

[Parser](#) - interface org.xml.sax.[Parser](#).

Deprecated. *This interface has been replaced by the SAX2 [XMLReader](#) interface, which includes Namespace support.*

[ParserAdapter](#) - class org.xml.sax.helpers.[ParserAdapter](#).

Adapt a SAX1 Parser as a SAX2 XMLReader.

[ParserAdapter\(\)](#) - Constructor for class org.xml.sax.helpers.[ParserAdapter](#)

Construct a new parser adapter.

[ParserAdapter\(Parser\)](#) - Constructor for class org.xml.sax.helpers.[ParserAdapter](#)

Construct a new parser adapter.

[ParserConfigurationException](#) - exception javax.xml.parsers.[ParserConfigurationException](#).

Indicates a serious configuration error.

[ParserConfigurationException\(\)](#) - Constructor for class javax.xml.parsers.[ParserConfigurationException](#)

Create a new `ParserConfigurationException` with no detail message.

[ParserConfigurationException\(String\)](#) - Constructor for class javax.xml.parsers.[ParserConfigurationException](#)

Create a new `ParserConfigurationException` with the `String` specified as an error message.

[ParserFactory](#) - class org.xml.sax.helpers.[ParserFactory](#).

Deprecated. *This class works with the deprecated [Parser](#) interface.*

[PI_DISABLE_OUTPUT_ESCAPING](#) - Static variable in interface javax.xml.transform.[Result](#)

The name of the processing instruction that is sent if the result tree disables output escaping.

[PI_ENABLE_OUTPUT_ESCAPING](#) - Static variable in interface javax.xml.transform.[Result](#)

The name of the processing instruction that is sent if the result tree enables output escaping at some point after having received a `PI_DISABLE_OUTPUT_ESCAPING` processing instruction.

[popContext\(\)](#) - Method in class org.xml.sax.helpers.[NamespaceSupport](#)

Revert to the previous Namespace context.

[printStackTrace\(\)](#) - Method in class javax.xml.transform.[TransformerException](#)

Print the the trace of methods from where the error originated.

[printStackTrace\(PrintStream\)](#) - Method in class javax.xml.transform.[TransformerException](#)

Print the the trace of methods from where the error originated.

[printStackTrace\(PrintWriter\)](#) - Method in class javax.xml.transform.[TransformerException](#)

Print the the trace of methods from where the error originated.

[PROCESSING_INSTRUCTION_NODE](#) - Static variable in interface org.w3c.dom.[Node](#)

The node is a `ProcessingInstruction`.

[ProcessingInstruction](#) - interface org.w3c.dom.[ProcessingInstruction](#).

The `ProcessingInstruction` interface represents a "processing instruction", used in XML as a way to keep processor-specific information in the text of the document.

[processingInstruction\(String, String\)](#) - Method in interface org.xml.sax.[ContentHandler](#)

Receive notification of a processing instruction.

[processingInstruction\(String, String\)](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Receive notification of a processing instruction.

[processingInstruction\(String, String\)](#) - Method in interface org.xml.sax.[DocumentHandler](#)

Deprecated. Receive notification of a processing instruction.

[processingInstruction\(String, String\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Adapt a SAX2 processing instruction event.

[processingInstruction\(String, String\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter a processing instruction event.

[processingInstruction\(String, String\)](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of a processing instruction.

[processingInstruction\(String, String\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Adapt a SAX1 processing instruction event.

[processName\(String, String\[\], boolean\)](#) - Method in class org.xml.sax.helpers.[NamespaceSupport](#)

Process a raw XML 1.0 name.

[pushContext\(\)](#) - Method in class org.xml.sax.helpers.[NamespaceSupport](#)

Start a new Namespace context.

R

[removeAttribute\(int\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Remove an attribute from the list.

[removeAttribute\(String\)](#) - Method in interface org.w3c.dom.[Element](#)

Removes an attribute by name.

[**removeAttribute\(String\)**](#) - Method in class org.xml.sax.helpers.[AttributeListImpl](#)

Deprecated. Remove an attribute from the list.

[**removeAttributeNode\(Attr\)**](#) - Method in interface org.w3c.dom.[Element](#)

Removes the specified attribute node.

[**removeAttributeNS\(String, String\)**](#) - Method in interface org.w3c.dom.[Element](#)

Removes an attribute by local name and namespace URI.

[**removeChild\(Node\)**](#) - Method in interface org.w3c.dom.[Node](#)

Removes the child node indicated by `oldChild` from the list of children, and returns it.

[**removeNamedItem\(String\)**](#) - Method in interface org.w3c.dom.[NamedNodeMap](#)

Removes a node specified by name.

[**removeNamedItemNS\(String, String\)**](#) - Method in interface org.w3c.dom.[NamedNodeMap](#)

Removes a node specified by local name and namespace URI.

[**replaceChild\(Node, Node\)**](#) - Method in interface org.w3c.dom.[Node](#)

Replaces the child node `oldChild` with `newChild` in the list of children, and returns the `oldChild` node.

[**replaceData\(int, int, String\)**](#) - Method in interface org.w3c.dom.[CharacterData](#)

Replace the characters starting at the specified 16-bit unit offset with the specified string.

[**reset\(\)**](#) - Method in class org.xml.sax.helpers.[NamespaceSupport](#)

Reset this Namespace support object for reuse.

[**resolve\(String, String\)**](#) - Method in interface javax.xml.transform.[URIResolver](#)

Called by the processor when it encounters an `xsl:include`, `xsl:import`, or `document()` function.

[**resolveEntity\(String, String\)**](#) - Method in interface org.xml.sax.[EntityResolver](#)

Allow the application to resolve external entities.

[**resolveEntity\(String, String\)**](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Resolve an external entity.

[**resolveEntity\(String, String\)**](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter an external entity resolution.

[**resolveEntity\(String, String\)**](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Resolve an external entity.

[**Result**](#) - interface javax.xml.transform.[Result](#).

An object that implements this interface contains the information needed to build a transformation result tree.

S

[SAXException](#) - exception org.xml.sax.[SAXException](#).

Encapsulate a general SAX error or warning.

[SAXException\(Exception\)](#) - Constructor for class org.xml.sax.[SAXException](#)

Create a new SAXException wrapping an existing exception.

[SAXException\(String\)](#) - Constructor for class org.xml.sax.[SAXException](#)

Create a new SAXException.

[SAXException\(String, Exception\)](#) - Constructor for class org.xml.sax.[SAXException](#)

Create a new SAXException from an existing exception.

[SAXNotRecognizedException](#) - exception org.xml.sax.[SAXNotRecognizedException](#).

Exception class for an unrecognized identifier.

[SAXNotRecognizedException\(String\)](#) - Constructor for class org.xml.sax.[SAXNotRecognizedException](#)

Construct a new exception with the given message.

[SAXNotSupportedException](#) - exception org.xml.sax.[SAXNotSupportedException](#).

Exception class for an unsupported operation.

[SAXNotSupportedException\(String\)](#) - Constructor for class org.xml.sax.[SAXNotSupportedException](#)

Construct a new exception with the given message.

[SAXParseException](#) - exception org.xml.sax.[SAXParseException](#).

Encapsulate an XML parse error or warning.

[SAXParseException\(String, Locator\)](#) - Constructor for class org.xml.sax.[SAXParseException](#)

Create a new SAXParseException from a message and a Locator.

[SAXParseException\(String, Locator, Exception\)](#) - Constructor for class org.xml.sax.[SAXParseException](#)

Wrap an existing exception in a SAXParseException.

[SAXParseException\(String, String, String, int, int\)](#) - Constructor for class org.xml.sax.[SAXParseException](#)

Create a new SAXParseException.

[SAXParseException\(String, String, String, int, int, Exception\)](#) - Constructor for class org.xml.sax.[SAXParseException](#)

Create a new SAXParseException with an embedded exception.

[SAXParser](#) - class javax.xml.parsers.[SAXParser](#).

Defines the API that wraps an [XMLReader](#) implementation class.

[SAXParser\(\)](#) - Constructor for class javax.xml.parsers.[SAXParser](#)

[SAXParserFactory](#) - class javax.xml.parsers.[SAXParserFactory](#).

Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents. An implementation of the SAXParserFactory class is *NOT* guaranteed to be thread safe.

[SAXParserFactory\(\)](#) - Constructor for class javax.xml.parsers.[SAXParserFactory](#)

[SAXResult](#) - class javax.xml.transform.sax.[SAXResult](#).

Acts as an holder for a transformation Result.

[SAXResult\(\)](#) - Constructor for class javax.xml.transform.sax.[SAXResult](#)

Zero-argument default constructor.

[SAXResult\(ContentHandler\)](#) - Constructor for class javax.xml.transform.sax.[SAXResult](#)

Create a SAXResult that targets a SAX2 [ContentHandler](#).

[SAXSource](#) - class javax.xml.transform.sax.[SAXSource](#).

Acts as an holder for SAX-style Source.

[SAXSource\(\)](#) - Constructor for class javax.xml.transform.sax.[SAXSource](#)

Zero-argument default constructor.

[SAXSource\(InputSource\)](#) - Constructor for class javax.xml.transform.sax.[SAXSource](#)

Create a SAXSource, using a SAX InputSource.

[SAXSource\(XMLReader, InputSource\)](#) - Constructor for class javax.xml.transform.sax.[SAXSource](#)

Create a SAXSource, using an [XMLReader](#) and a SAX InputSource.

[SAXTransformerFactory](#) - class javax.xml.transform.sax.[SAXTransformerFactory](#).

This class extends TransformerFactory to provide SAX-specific factory methods.

[SAXTransformerFactory\(\)](#) - Constructor for class javax.xml.transform.sax.[SAXTransformerFactory](#)

The default constructor is protected on purpose.

[setAttribute\(int, String, String, String, String, String\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Set an attribute in the list.

[setAttribute\(String, Object\)](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Allows the user to set specific attributes on the underlying implementation.

[setAttribute\(String, Object\)](#) - Method in class javax.xml.transform.[TransformerFactory](#)

Allows the user to set specific attributes on the underlying implementation.

[setAttribute\(String, String\)](#) - Method in interface org.w3c.dom.[Element](#)

Adds a new attribute.

[setAttributeList\(AttributeList\)](#) - Method in class org.xml.sax.helpers.[AttributeListImpl](#)

Deprecated. Set the attribute list, discarding previous contents.

[setAttributeNode\(Attr\)](#) - Method in interface org.w3c.dom.[Element](#)

Adds a new attribute node.

[setAttributeNodeNS\(Attr\)](#) - Method in interface org.w3c.dom.[Element](#)

Adds a new attribute.

[setAttributeNS\(String, String, String\)](#) - Method in interface org.w3c.dom.[Element](#)

Adds a new attribute.

[setAttributes\(Attributes\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Copy an entire Attributes object.

[setByteStream\(InputStream\)](#) - Method in class org.xml.sax.[InputSource](#)

Set the byte stream for this input source.

[setCharacterStream\(Reader\)](#) - Method in class org.xml.sax.[InputSource](#)

Set the character stream for this input source.

[setCoalescing\(boolean\)](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Specifies that the parser produced by this code will convert CDATA nodes to Text nodes and append it to the adjacent (if any) text node.

[setColumnNumber\(int\)](#) - Method in class org.xml.sax.helpers.[LocatorImpl](#)

Set the column number for this locator (1-based).

[setContentHandler\(ContentHandler\)](#) - Method in interface org.xml.sax.[XMLReader](#)

Allow an application to register a content event handler.

[setContentHandler\(ContentHandler\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Set the content event handler.

[setContentHandler\(ContentHandler\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Set the content handler.

[setData\(String\)](#) - Method in interface org.w3c.dom.[ProcessingInstruction](#)

[setData\(String\)](#) - Method in interface org.w3c.dom.[CharacterData](#)

[setDocumentHandler\(DocumentHandler\)](#) - Method in interface org.xml.sax.[Parser](#)

Deprecated. Allow an application to register a document event handler.

[setDocumentHandler\(DocumentHandler\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Register the SAX1 document event handler.

[setDocumentLocator\(Locator\)](#) - Method in interface org.xml.sax.[ContentHandler](#)

Receive an object for locating the origin of SAX document events.

[setDocumentLocator\(Locator\)](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Receive a Locator object for document events.

[setDocumentLocator\(Locator\)](#) - Method in interface org.xml.sax.[DocumentHandler](#)

Deprecated. Receive an object for locating the origin of SAX document events.

[setDocumentLocator\(Locator\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Set a document locator.

[setDocumentLocator\(Locator\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter a new document locator event.

[setDocumentLocator\(Locator\)](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive a Locator object for document events.

[setDocumentLocator\(Locator\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Adapt a SAX1 document locator event.

[setDTDHandler\(DTDHandler\)](#) - Method in interface org.xml.sax.[Parser](#)

Deprecated. Allow an application to register a DTD event handler.

[setDTDHandler\(DTDHandler\)](#) - Method in interface org.xml.sax.[XMLReader](#)

Allow an application to register a DTD event handler.

[setDTDHandler\(DTDHandler\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Register the DTD event handler.

[setDTDHandler\(DTDHandler\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Set the DTD event handler.

[setDTDHandler\(DTDHandler\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Set the DTD handler.

[setEncoding\(String\)](#) - Method in class org.xml.sax.[InputSource](#)

Set the character encoding, if known.

[setEntityResolver\(EntityResolver\)](#) - Method in class javax.xml.parsers.[DocumentBuilder](#)

Specify the [EntityResolver](#) to be used to resolve entities present in the XML document to be parsed.

[setEntityResolver\(EntityResolver\)](#) - Method in interface org.xml.sax.[Parser](#)

Deprecated. Allow an application to register a custom entity resolver.

[setEntityResolver\(EntityResolver\)](#) - Method in interface org.xml.sax.[XMLReader](#)

Allow an application to register an entity resolver.

[setEntityResolver\(EntityResolver\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Register the entity resolver.

[setEntityResolver\(EntityResolver\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Set the entity resolver.

[setEntityResolver\(EntityResolver\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Set the entity resolver.

[setErrorHandler\(ErrorHandler\)](#) - Method in class javax.xml.parsers.[DocumentBuilder](#)

Specify the [ErrorHandler](#) to be used to report errors present in the XML document to be parsed.

[setErrorHandler\(ErrorHandler\)](#) - Method in interface org.xml.sax.[Parser](#)

Deprecated. Allow an application to register an error event handler.

[setErrorHandler\(ErrorHandler\)](#) - Method in interface org.xml.sax.[XMLReader](#)

Allow an application to register an error event handler.

[setErrorHandler\(ErrorHandler\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Register the error event handler.

[setErrorHandler\(ErrorHandler\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Set the error event handler.

[setErrorHandler\(ErrorHandler\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Set the error handler.

[setErrorListener\(ErrorListener\)](#) - Method in class javax.xml.transform.[Transformer](#)

Set the error event listener in effect for the transformation.

[setErrorListener\(ErrorListener\)](#) - Method in class javax.xml.transform.[TransformerFactory](#)

Set the error event listener for the TransformerFactory, which is used for the processing of transformation instructions, and not for the transformation itself.

[setExpandEntityReferences\(boolean\)](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Specifies that the parser produced by this code will expand entity reference nodes.

[setFeature\(String, boolean\)](#) - Method in class javax.xml.parsers.[SAXParserFactory](#)

Sets the particular feature in the underlying implementation of org.xml.sax.XMLReader.

[setFeature\(String, boolean\)](#) - Method in interface org.xml.sax.[XMLReader](#)

Set the state of a feature.

[setFeature\(String, boolean\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Set the state of a feature.

[setFeature\(String, boolean\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Set a feature for the parser.

[setHandler\(ContentHandler\)](#) - Method in class javax.xml.transform.sax.[SAXResult](#)

Set the target to be a SAX2 [ContentHandler](#).

[setIgnoringComments\(boolean\)](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Specifies that the parser produced by this code will ignore comments.

[setIgnoringElementContentWhitespace\(boolean\)](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Specifies that the parsers created by this factory must eliminate whitespace in element content (sometimes known loosely as 'ignorable whitespace') when parsing XML documents (see XML Rec 2.10).

[setInputSource\(InputSource\)](#) - Method in class javax.xml.transform.sax.[SAXSource](#)

Set the SAX InputSource to be used for the Source.

[setInputStream\(InputStream\)](#) - Method in class javax.xml.transform.stream.[StreamSource](#)

Set the byte stream to be used as input.

[setLexicalHandler\(LexicalHandler\)](#) - Method in class javax.xml.transform.sax.[SAXResult](#)

Set the SAX2 [LexicalHandler](#) for the output.

[setLineNumber\(int\)](#) - Method in class org.xml.sax.helpers.[LocatorImpl](#)

Set the line number for this locator (1-based).

[setLocale\(Locale\)](#) - Method in interface org.xml.sax.[Parser](#)

Deprecated. Allow an application to request a locale for errors and warnings.

[setLocale\(Locale\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Set the locale for error reporting.

[setLocalName\(int, String\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Set the local name of a specific attribute.

[setLocator\(SourceLocator\)](#) - Method in class javax.xml.transform.[TransformerException](#)

Method setLocator sets an instance of a SourceLocator object that specifies where an error occurred.

[setNamedItem\(Node\)](#) - Method in interface org.w3c.dom.[NamedNodeMap](#)

Adds a node using its nodeName attribute.

[setNamedItemNS\(Node\)](#) - Method in interface org.w3c.dom.[NamedNodeMap](#)

Adds a node using its namespaceURI and localName.

[setNamespaceAware\(boolean\)](#) - Method in class javax.xml.parsers.[SAXParserFactory](#)

Specifies that the parser produced by this code will provide support for XML namespaces.

[setNamespaceAware\(boolean\)](#) - Method in class javax.xml.parsers.[DocumentBuilderFactory](#)

Specifies that the parser produced by this code will provide support for XML namespaces.

[setNode\(Node\)](#) - Method in class javax.xml.transform.dom.[DOMSource](#)

Set the node that will represents a Source DOM tree.

[setNode\(Node\)](#) - Method in class javax.xml.transform.dom.[DOMResult](#)

Set the node that will contain the result DOM tree.

[setNodeValue\(String\)](#) - Method in interface org.w3c.dom.[Node](#)

[setOutputProperties\(Properties\)](#) - Method in class javax.xml.transform.[Transformer](#)

Set the output properties for the transformation.

[**setOutputProperty\(String, String\)**](#) - Method in class javax.xml.transform.[Transformer](#)

Set an output property that will be in effect for the transformation.

[**setOutputStream\(OutputStream\)**](#) - Method in class javax.xml.transform.stream.[StreamResult](#)

Set the ByteStream that is to be written to.

[**setParameter\(String, Object\)**](#) - Method in class javax.xml.transform.[Transformer](#)

Add a parameter for the transformation.

[**setParent\(XMLReader\)**](#) - Method in interface org.xml.sax.[XMLFilter](#)

Set the parent reader.

[**setParent\(XMLReader\)**](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Set the parent reader.

[**setPrefix\(String\)**](#) - Method in interface org.w3c.dom.[Node](#)

[**setProperty\(String, Object\)**](#) - Method in class javax.xml.parsers.[SAXParser](#)

Sets the particular property in the underlying implementation of [XMLReader](#).

[**setProperty\(String, Object\)**](#) - Method in interface org.xml.sax.[XMLReader](#)

Set the value of a property.

[**setProperty\(String, Object\)**](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Set the value of a property.

[**setProperty\(String, Object\)**](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Set a parser property.

[**setPublicId\(String\)**](#) - Method in class javax.xml.transform.stream.[StreamSource](#)

Set the public identifier for this Source.

[**setPublicId\(String\)**](#) - Method in class org.xml.sax.[InputSource](#)

Set the public identifier for this input source.

[**setPublicId\(String\)**](#) - Method in class org.xml.sax.helpers.[LocatorImpl](#)

Set the public identifier for this locator.

[**setQName\(int, String\)**](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Set the qualified name of a specific attribute.

[**setReader\(Reader\)**](#) - Method in class javax.xml.transform.stream.[StreamSource](#)

Set the input to be a character reader.

[**setResult\(Result\)**](#) - Method in interface javax.xml.transform.sax.[TransformerHandler](#)

Enables the user of the TransformerHandler to set the to set the Result for the transformation.

[**setSystemId\(File\)**](#) - Method in class javax.xml.transform.stream.[StreamResult](#)

Set the system ID from a File reference.

[setSystemId\(File\)](#) - Method in class javax.xml.transform.stream.[StreamSource](#)

Set the system ID from a File reference.

[setSystemId\(String\)](#) - Method in interface javax.xml.transform.[Result](#)

Set the system identifier for this Result.

[setSystemId\(String\)](#) - Method in interface javax.xml.transform.[Source](#)

Set the system identifier for this Source.

[setSystemId\(String\)](#) - Method in class javax.xml.transform.dom.[DOMSource](#)

Set the base ID (URL or system ID) from where URLs will be resolved.

[setSystemId\(String\)](#) - Method in class javax.xml.transform.dom.[DOMResult](#)

Method setSystemId Set the systemID that may be used in association with the node.

[setSystemId\(String\)](#) - Method in interface javax.xml.transform.sax.[TransformerHandler](#)

Set the base ID (URI or system ID) from where relative URLs will be resolved.

[setSystemId\(String\)](#) - Method in interface javax.xml.transform.sax.[TemplatesHandler](#)

Set the base ID (URI or system ID) for the Templates object created by this builder.

[setSystemId\(String\)](#) - Method in class javax.xml.transform.sax.[SAXResult](#)

Method setSystemId Set the systemID that may be used in association with the [ContentHandler](#).

[setSystemId\(String\)](#) - Method in class javax.xml.transform.sax.[SAXSource](#)

Set the system identifier for this Source.

[setSystemId\(String\)](#) - Method in class javax.xml.transform.stream.[StreamResult](#)

Set the systemID that may be used in association with the byte or character stream, or, if neither is set, use this value as a writeable URI (probably a file name).

[setSystemId\(String\)](#) - Method in class javax.xml.transform.stream.[StreamSource](#)

Set the system identifier for this Source.

[setSystemId\(String\)](#) - Method in class org.xml.sax.[InputSource](#)

Set the system identifier for this input source.

[setSystemId\(String\)](#) - Method in class org.xml.sax.helpers.[LocatorImpl](#)

Set the system identifier for this locator.

[setType\(int, String\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Set the type of a specific attribute.

[setURI\(int, String\)](#) - Method in class org.xml.sax.helpers.[AttributesImpl](#)

Set the Namespace URI of a specific attribute.

[setURIResolver\(URIResolver\)](#) - Method in class javax.xml.transform.[Transformer](#)

Set an object that will be used to resolve URIs used in document().

[setURIResolver\(URIResolver\)](#) - Method in class javax.xml.transform.[TransformerFactory](#)

Set an object that is used by default during the transformation to resolve URIs used in `xsl:import`, or `xsl:include`.

[setValidating\(boolean\)](#) - Method in class `javax.xml.parsers.SAXParserFactory`

Specifies that the parser produced by this code will validate documents as they are parsed.

[setValidating\(boolean\)](#) - Method in class `javax.xml.parsers.DocumentBuilderFactory`

Specifies that the parser produced by this code will validate documents as they are parsed.

[setValue\(int, String\)](#) - Method in class `org.xml.sax.helpers.AttributesImpl`

Set the value of a specific attribute.

[setValue\(String\)](#) - Method in interface `org.w3c.dom.Attr`

[setWriter\(Writer\)](#) - Method in class `javax.xml.transform.stream.StreamResult`

Set the writer that is to receive the result.

[setXMLReader\(XMLReader\)](#) - Method in class `javax.xml.transform.sax.SAXSource`

Set the XMLReader to be used for the Source.

[skippedEntity\(String\)](#) - Method in interface `org.xml.sax.ContentHandler`

Receive notification of a skipped entity.

[skippedEntity\(String\)](#) - Method in class `org.xml.sax.helpers.XMLReaderAdapter`

Adapt a SAX2 skipped entity event.

[skippedEntity\(String\)](#) - Method in class `org.xml.sax.helpers.XMLFilterImpl`

Filter a skipped entity event.

[skippedEntity\(String\)](#) - Method in class `org.xml.sax.helpers.DefaultHandler`

Receive notification of a skipped entity.

[Source](#) - interface `javax.xml.transform.Source`.

An object that implements this interface contains the information needed to act as source input (XML source or transformation instructions).

[SourceLocator](#) - interface `javax.xml.transform.SourceLocator`.

This interface is primarily for the purposes of reporting where an error occurred in the XML source or transformation instructions.

[sourceToInputSource\(Source\)](#) - Static method in class `javax.xml.transform.sax.SAXSource`

Attempt to obtain a SAX InputSource object from a TrAX Source object.

[splitText\(int\)](#) - Method in interface `org.w3c.dom.Text`

Breaks this node into two nodes at the specified `offset`, keeping both in the tree as siblings.

[STANDALONE](#) - Static variable in class `javax.xml.transform.OutputKeys`

`standalone = "yes" | "no"`.

[startCDATA\(\)](#) - Method in interface `org.xml.sax.ext.LexicalHandler`

Report the start of a CDATA section.

[startDocument\(\)](#) - Method in interface org.xml.sax.[ContentHandler](#)

Receive notification of the beginning of a document.

[startDocument\(\)](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Receive notification of the beginning of the document.

[startDocument\(\)](#) - Method in interface org.xml.sax.[DocumentHandler](#)

Deprecated. Receive notification of the beginning of a document.

[startDocument\(\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Start document event.

[startDocument\(\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter a start document event.

[startDocument\(\)](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of the beginning of the document.

[startDocument\(\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Adapt a SAX1 start document event.

[startDTD\(String, String, String\)](#) - Method in interface org.xml.sax.ext.[LexicalHandler](#)

Report the start of DTD declarations, if any.

[startElement\(String, AttributeList\)](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Receive notification of the start of an element.

[startElement\(String, AttributeList\)](#) - Method in interface org.xml.sax.[DocumentHandler](#)

Deprecated. Receive notification of the beginning of an element.

[startElement\(String, AttributeList\)](#) - Method in class org.xml.sax.helpers.[ParserAdapter](#)

Adapt a SAX1 startElement event.

[startElement\(String, String, String, Attributes\)](#) - Method in interface org.xml.sax.[ContentHandler](#)

Receive notification of the beginning of an element.

[startElement\(String, String, String, Attributes\)](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Adapt a SAX2 start element event.

[startElement\(String, String, String, Attributes\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter a start element event.

[startElement\(String, String, String, Attributes\)](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of the start of an element.

[startEntity\(String\)](#) - Method in interface org.xml.sax.ext.[LexicalHandler](#)

Report the beginning of some internal and external XML entities.

[startPrefixMapping\(String, String\)](#) - Method in interface org.xml.sax.[ContentHandler](#)

Begin the scope of a prefix-URI Namespace mapping.

[**startPrefixMapping\(String, String\)**](#) - Method in class org.xml.sax.helpers.[XMLReaderAdapter](#)

Adapt a SAX2 start prefix mapping event.

[**startPrefixMapping\(String, String\)**](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter a start Namespace prefix mapping event.

[**startPrefixMapping\(String, String\)**](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of the start of a Namespace mapping.

[**StreamResult**](#) - class javax.xml.transform.stream.[StreamResult](#).

Acts as an holder for a transformation result, which may be XML, plain Text, HTML, or some other form of markup.

[**StreamResult\(\)**](#) - Constructor for class javax.xml.transform.stream.[StreamResult](#)

Zero-argument default constructor.

[**StreamResult\(File\)**](#) - Constructor for class javax.xml.transform.stream.[StreamResult](#)

Construct a StreamResult from a File.

[**StreamResult\(OutputStream\)**](#) - Constructor for class javax.xml.transform.stream.[StreamResult](#)

Construct a StreamResult from a byte stream.

[**StreamResult\(String\)**](#) - Constructor for class javax.xml.transform.stream.[StreamResult](#)

Construct a StreamResult from a URL.

[**StreamResult\(Writer\)**](#) - Constructor for class javax.xml.transform.stream.[StreamResult](#)

Construct a StreamResult from a character stream.

[**StreamSource**](#) - class javax.xml.transform.stream.[StreamSource](#).

Acts as an holder for a transformation Source in the form of a stream of XML markup.

[**StreamSource\(\)**](#) - Constructor for class javax.xml.transform.stream.[StreamSource](#)

Zero-argument default constructor.

[**StreamSource\(File\)**](#) - Constructor for class javax.xml.transform.stream.[StreamSource](#)

Construct a StreamSource from a File.

[**StreamSource\(InputStream\)**](#) - Constructor for class javax.xml.transform.stream.[StreamSource](#)

Construct a StreamSource from a byte stream.

[**StreamSource\(InputStream, String\)**](#) - Constructor for class javax.xml.transform.stream.[StreamSource](#)

Construct a StreamSource from a byte stream.

[**StreamSource\(Reader\)**](#) - Constructor for class javax.xml.transform.stream.[StreamSource](#)

Construct a StreamSource from a character reader.

[**StreamSource\(Reader, String\)**](#) - Constructor for class javax.xml.transform.stream.[StreamSource](#)

Construct a StreamSource from a character reader.

[**StreamSource\(String\)**](#) - Constructor for class javax.xml.transform.stream.[StreamSource](#)

Construct a `StreamSource` from a URL.

[substringData\(int, int\)](#) - Method in interface `org.w3c.dom.CharacterData`

Extracts a range of data from the node.

[SYNTAX_ERR](#) - Static variable in class `org.w3c.dom.DOMException`

If an invalid or illegal string is specified.

T

[Templates](#) - interface `javax.xml.transform.Templates`.

An object that implements this interface is the runtime representation of processed transformation instructions.

[TemplatesHandler](#) - interface `javax.xml.transform.sax.TemplatesHandler`.

A SAX ContentHandler that may be used to process SAX parse events (parsing transformation instructions) into a `Templates` object.

[Text](#) - interface `org.w3c.dom.Text`.

The `Text` interface inherits from `CharacterData` and represents the textual content (termed character data in XML) of an `Element` or `Attr`.

[TEXT_NODE](#) - Static variable in interface `org.w3c.dom.Node`

The node is a `Text` node.

[toString\(\)](#) - Method in class `org.xml.sax.SAXException`

Override `toString` to pick up any embedded exception.

[transform\(Source, Result\)](#) - Method in class `javax.xml.transform.Transformer`

Process the source tree to the output result.

[Transformer](#) - class `javax.xml.transform.Transformer`.

An instance of this abstract class can transform a source tree into a result tree.

[Transformer\(\)](#) - Constructor for class `javax.xml.transform.Transformer`

Default constructor is protected on purpose.

[TransformerConfigurationException](#) - exception

`javax.xml.transform.TransformerConfigurationException`.

Indicates a serious configuration error.

[TransformerConfigurationException\(\)](#) - Constructor for class

`javax.xml.transform.TransformerConfigurationException`

Create a new `TransformerConfigurationException` with no detail message.

[TransformerConfigurationException\(String\)](#) - Constructor for class

`javax.xml.transform.TransformerConfigurationException`

Create a new `TransformerConfigurationException` with the `String` specified as an

error message.

[TransformerConfigurationException\(String, SourceLocator\)](#) - Constructor for class `javax.xml.transform.TransformerConfigurationException`

Create a new `TransformerConfigurationException` from a message and a Locator.

[TransformerConfigurationException\(String, SourceLocator, Throwable\)](#) - Constructor for class `javax.xml.transform.TransformerConfigurationException`

Wrap an existing exception in a `TransformerConfigurationException`.

[TransformerConfigurationException\(String, Throwable\)](#) - Constructor for class `javax.xml.transform.TransformerConfigurationException`

Create a new `TransformerConfigurationException` with the given `Exception` base cause and detail message.

[TransformerConfigurationException\(Throwable\)](#) - Constructor for class `javax.xml.transform.TransformerConfigurationException`

Create a new `TransformerConfigurationException` with a given `Exception` base cause of the error.

[TransformerException](#) - exception `javax.xml.transform.TransformerException`.

This class specifies an exceptional condition that occurred during the transformation process.

[TransformerException\(String\)](#) - Constructor for class `javax.xml.transform.TransformerException`

Create a new `TransformerException`.

[TransformerException\(String, SourceLocator\)](#) - Constructor for class `javax.xml.transform.TransformerException`

Create a new `TransformerException` from a message and a Locator.

[TransformerException\(String, SourceLocator, Throwable\)](#) - Constructor for class `javax.xml.transform.TransformerException`

Wrap an existing exception in a `TransformerException`.

[TransformerException\(String, Throwable\)](#) - Constructor for class `javax.xml.transform.TransformerException`

Wrap an existing exception in a `TransformerException`.

[TransformerException\(Throwable\)](#) - Constructor for class `javax.xml.transform.TransformerException`

Create a new `TransformerException` wrapping an existing exception.

[TransformerFactory](#) - class `javax.xml.transform.TransformerFactory`.

A `TransformerFactory` instance can be used to create [Transformer](#) and [Templates](#) objects.

[TransformerFactory\(\)](#) - Constructor for class `javax.xml.transform.TransformerFactory`

Default constructor is protected on purpose.

[TransformerFactoryConfigurationError](#) - error

`javax.xml.transform.TransformerFactoryConfigurationError`.

Thrown when a problem with configuration with the Transformer Factories exists.

[TransformerFactoryConfigurationError\(\)](#) - Constructor for class

javax.xml.transform.[TransformerFactoryConfigurationError](#)

Create a new TransformerFactoryConfigurationError with no detail message.

[TransformerFactoryConfigurationError\(Exception\)](#) - Constructor for class

javax.xml.transform.[TransformerFactoryConfigurationError](#)

Create a new TransformerFactoryConfigurationError with a given Exception base cause of the error.

[TransformerFactoryConfigurationError\(Exception, String\)](#) - Constructor for class

javax.xml.transform.[TransformerFactoryConfigurationError](#)

Create a new TransformerFactoryConfigurationError with the given Exception base cause and detail message.

[TransformerFactoryConfigurationError\(String\)](#) - Constructor for class

javax.xml.transform.[TransformerFactoryConfigurationError](#)

Create a new TransformerFactoryConfigurationError with the String specified as an error message.

[TransformerHandler](#) - interface javax.xml.transform.sax.[TransformerHandler](#).

A TransformerHandler listens for SAX ContentHandler parse events and transforms them to a Result.

U

[unparsedEntityDecl\(String, String, String, String\)](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Receive notification of an unparsed entity declaration.

[unparsedEntityDecl\(String, String, String, String\)](#) - Method in interface org.xml.sax.[DTDHandler](#)

Receive notification of an unparsed entity declaration event.

[unparsedEntityDecl\(String, String, String, String\)](#) - Method in class

org.xml.sax.helpers.[XMLFilterImpl](#)

Filter an unparsed entity declaration event.

[unparsedEntityDecl\(String, String, String, String\)](#) - Method in class

org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of an unparsed entity declaration.

[URIResolver](#) - interface javax.xml.transform.[URIResolver](#).

An object that implements this interface that can be called by the processor to turn a URI used in document(), xsl:import, or xsl:include into a Source object.

V

[VERSION](#) - Static variable in class javax.xml.transform.[OutputKeys](#)

version = nmtoken.

W

[warning\(SAXParseException\)](#) - Method in class org.xml.sax.[HandlerBase](#)

Deprecated. Receive notification of a parser warning.

[warning\(SAXParseException\)](#) - Method in interface org.xml.sax.[ErrorHandler](#)

Receive notification of a warning.

[warning\(SAXParseException\)](#) - Method in class org.xml.sax.helpers.[XMLFilterImpl](#)

Filter a warning event.

[warning\(SAXParseException\)](#) - Method in class org.xml.sax.helpers.[DefaultHandler](#)

Receive notification of a parser warning.

[warning\(TransformerException\)](#) - Method in interface javax.xml.transform.[ErrorListener](#)

Receive notification of a warning.

[WRONG_DOCUMENT_ERR](#) - Static variable in class org.w3c.dom.[DOMException](#)

If a node is used in a different document than the one that created it (that doesn't support it)

X

[XMLFilter](#) - interface org.xml.sax.[XMLFilter](#).

Interface for an XML filter.

[XMLFilterImpl](#) - class org.xml.sax.helpers.[XMLFilterImpl](#).

Base class for deriving an XML filter.

[XMLFilterImpl\(\)](#) - Constructor for class org.xml.sax.helpers.[XMLFilterImpl](#)

Construct an empty XML filter, with no parent.

[XMLFilterImpl\(XMLReader\)](#) - Constructor for class org.xml.sax.helpers.[XMLFilterImpl](#)

Construct an XML filter with the specified parent.

[XMLNS](#) - Static variable in class org.xml.sax.helpers.[NamespaceSupport](#)

The XML Namespace as a constant.

[XMLReader](#) - interface org.xml.sax.[XMLReader](#).

Interface for reading an XML document using callbacks.

[**XMLReaderAdapter**](#) - class org.xml.sax.helpers.[**XMLReaderAdapter**](#).

Adapt a SAX2 XMLReader as a SAX1 Parser.

[**XMLReaderAdapter\(\)**](#) - Constructor for class org.xml.sax.helpers.[**XMLReaderAdapter**](#)

Create a new adapter.

[**XMLReaderAdapter\(XMLReader\)**](#) - Constructor for class org.xml.sax.helpers.[**XMLReaderAdapter**](#)

Create a new adapter.

[**XMLReaderFactory**](#) - class org.xml.sax.helpers.[**XMLReaderFactory**](#).

Factory for creating an XML reader.

[A](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [L](#) [M](#) [N](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#)

[**Overview**](#) [Package](#) [Class](#) [Use](#) [**Tree**](#) [**Deprecated**](#) [Index](#) [**Help**](#)

[PREV](#) [NEXT](#)

[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV PACKAGE](#)
[NEXT PACKAGE](#)
[FRAMES](#)
[NO FRAMES](#)

Package javax.xml.transform.sax

This package implements SAX2-specific transformation APIs.

See:

[Description](#)

Interface Summary

<i>TemplatesHandler</i>	A SAX ContentHandler that may be used to process SAX parse events (parsing transformation instructions) into a Templates object.
<i>TransformerHandler</i>	A TransformerHandler listens for SAX ContentHandler parse events and transforms them to a Result.

Class Summary

<i>SAXResult</i>	Acts as an holder for a transformation Result.
<i>SAXSource</i>	Acts as an holder for SAX-style Source.
<i>SAXTransformerFactory</i>	This class extends TransformerFactory to provide SAX-specific factory methods.

Package javax.xml.transform.sax Description

This package implements SAX2-specific transformation APIs. It provides classes which allow input from [ContentHandler](#) events, and also classes that produce org.xml.sax.ContentHandler events. It also provides methods to set the input source as an [XMLReader](#), or to use a [InputSource](#) as the source. It also allows the creation of a [XMLFilter](#), which enables transformations to "pull" from other transformations, and lets the transformer to be used polymorphically as an [XMLReader](#).

The [SAXSource](#) class allows the setting of an [XMLReader](#) to be used for "pulling" parse events, and an [InputSource](#) that may be used to specify the SAX source.

The [SAXResult](#) class allows the setting of a [ContentHandler](#) to be the receiver of SAX2 events from the transformation.

The [SAXTransformerFactory](#) extends [TransformerFactory](#) to provide factory methods for creating [TemplatesHandler](#), [TransformerHandler](#), and [XMLReader](#) instances.

To obtain a [SAXTransformerFactory](#), the caller must cast the [TransformerFactory](#) instance returned from [TransformerFactory.newInstance\(\)](#).

The [TransformerHandler](#) interface allows a transformation to be created from SAX2 parse events, which is a "push" model rather than the "pull" model that normally occurs for a transformation. Normal parse events are received through the [ContentHandler](#) interface, lexical events such as startCDATA and endCDATA are received through the [LexicalHandler](#) interface, and events that signal the start or end of disabling output escaping are received via [ContentHandler.processingInstruction\(java.lang.String, java.lang.String\)](#), with the target parameter being [Result.PI_DISABLE_OUTPUT_ESCAPING](#) and [Result.PI_ENABLE_OUTPUT_ESCAPING](#). If parameters, output properties, or other features need to be set on the Transformer handler, a [Transformer](#) reference will need to be obtained from [TransformerHandler.getTransformer\(\)](#), and the methods invoked from that reference.

The [TemplatesHandler](#) interface allows the creation of [Templates](#) objects from SAX2 parse events. Once the [ContentHandler](#) events are complete, the Templates object may be obtained from [TemplatesHandler.getTemplates\(\)](#). Note that [TemplatesHandler.setSystemId\(java.lang.String\)](#) should normally be called in order to establish a base system ID from which relative URLs may be resolved.

The [SAXTransformerFactory.newXMLFilter\(javax.xml.transform.Source\)](#) method allows the creation of a [XMLFilter](#), which encapsulates the SAX2 notion of a "pull" transformation. The following illustrates several transformations chained together. Each filter points to a parent [XMLReader](#), and the final transformation is caused by invoking [XMLReader.parse\(org.xml.sax.InputSource\)](#) on the final reader in the chain.

[Overview](#) **Package** Class [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV PACKAGE](#) [NEXT PACKAGE](#)

[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV PACKAGE](#)
[NEXT PACKAGE](#)
[FRAMES](#)
[NO FRAMES](#)

Package javax.xml.transform.stream

This package implements stream- and URI- specific transformation APIs.

See:

[Description](#)

Class Summary

[StreamResult](#)

Acts as an holder for a transformation result, which may be XML, plain Text, HTML, or some other form of markup.

[StreamSource](#)

Acts as an holder for a transformation Source in the form of a stream of XML markup.

Package javax.xml.transform.stream Description

This package implements stream- and URI- specific transformation APIs.

The [StreamSource](#) class provides methods for specifying InputStream input, Reader input, and URL input in the form of strings. Even if an input stream or reader is specified as the source, [StreamSource.setSystemId\(java.lang.String\)](#) should still be called, so that the transformer can know from where it should resolve relative URIs. The public identifier is always optional: if the application writer includes one, it will be provided as part of the [SourceLocator](#) information.

The [StreamResult](#) class provides methods for specifying OutputStream, Writer, or an output system ID, as the output of the transformation result.

Normally streams should be used rather than readers or writers, for both the Source and Result, since readers and writers already have the encoding established to and from the internal Unicode format. However, there are times when it is useful to write to a character stream, such as when using a StringWriter in order to write to a String, or in the case of reading source XML from a StringReader.

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV PACKAGE](#)
[NEXT PACKAGE](#)
[FRAMES](#)
[NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV PACKAGE](#)
[NEXT PACKAGE](#)
[FRAMES](#)
[NO FRAMES](#)

Package org.w3c.dom

Interface Summary

<u><i>Attr</i></u>	The <code>Attr</code> interface represents an attribute in an <code>Element</code> object.
<u><i>CDATASection</i></u>	CDATA sections are used to escape blocks of text containing characters that would otherwise be regarded as markup.
<u><i>CharacterData</i></u>	The <code>CharacterData</code> interface extends <code>Node</code> with a set of attributes and methods for accessing character data in the DOM.
<u><i>Comment</i></u>	This interface inherits from <code>CharacterData</code> and represents the content of a comment, i.e., all the characters between the starting ' <code><!--</code> ' and ending ' <code>--></code> '.
<u><i>Document</i></u>	The <code>Document</code> interface represents the entire HTML or XML document.
<u><i>DocumentFragment</i></u>	<code>DocumentFragment</code> is a "lightweight" or "minimal" <code>Document</code> object.
<u><i>DocumentType</i></u>	Each <code>Document</code> has a <code>doctype</code> attribute whose value is either <code>null</code> or a <code>DocumentType</code> object.
<u><i>DOMImplementation</i></u>	The <code>DOMImplementation</code> interface provides a number of methods for performing operations that are independent of any particular instance of the document object model.
<u><i>Element</i></u>	The <code>Element</code> interface represents an element in an HTML or XML document.
<u><i>Entity</i></u>	This interface represents an entity, either parsed or unparsed, in an XML document.
<u><i>EntityReference</i></u>	<code>EntityReference</code> objects may be inserted into the structure model when an entity reference is in the source document, or when the user wishes to insert an entity reference.
<u><i>NamedNodeMap</i></u>	Objects implementing the <code>NamedNodeMap</code> interface are used to represent collections of nodes that can be accessed by name.
<u><i>Node</i></u>	The <code>Node</code> interface is the primary datatype for the entire Document Object Model.
<u><i>NodeList</i></u>	The <code>NodeList</code> interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented.
<u><i>Notation</i></u>	This interface represents a notation declared in the DTD.

<u><i>ProcessingInstruction</i></u>	The <code>ProcessingInstruction</code> interface represents a "processing instruction", used in XML as a way to keep processor-specific information in the text of the document.
<u><i>Text</i></u>	The <code>Text</code> interface inherits from <code>CharacterData</code> and represents the textual content (termed character data in XML) of an <code>Element</code> or <code>Attr</code> .

Exception Summary

<u><i>DOMException</i></u>	DOM operations only raise exceptions in "exceptional" circumstances, i.e., when an operation is impossible to perform (either for logical reasons, because data is lost, or because the implementation has become unstable).
--	--

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)

[PREV PACKAGE](#)
[NEXT PACKAGE](#)

[FRAMES](#)
[NO FRAMES](#)

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV PACKAGE](#)
[NEXT PACKAGE](#)
[FRAMES](#)
[NO FRAMES](#)

Package org.xml.sax

Interface Summary

<u>AttributeList</u>	Deprecated. <i>This interface has been replaced by the SAX2 <u>Attributes</u> interface, which includes Namespace support.</i>
<u>Attributes</u>	Interface for a list of XML attributes.
<u>ContentHandler</u>	Receive notification of the logical content of a document.
<u>DocumentHandler</u>	Deprecated. <i>This interface has been replaced by the SAX2 <u>ContentHandler</u> interface, which includes Namespace support.</i>
<u>DTDHandler</u>	Receive notification of basic DTD-related events.
<u>EntityResolver</u>	Basic interface for resolving entities.
<u>ErrorHandler</u>	Basic interface for SAX error handlers.
<u>Locator</u>	Interface for associating a SAX event with a document location.
<u>Parser</u>	Deprecated. <i>This interface has been replaced by the SAX2 <u>XMLReader</u> interface, which includes Namespace support.</i>
<u>XMLFilter</u>	Interface for an XML filter.
<u>XMLReader</u>	Interface for reading an XML document using callbacks.

Class Summary

<u>HandlerBase</u>	Deprecated. <i>This class works with the deprecated <u>DocumentHandler</u> interface.</i>
<u>InputSource</u>	A single input source for an XML entity.

Exception Summary

<u>SAXException</u>	Encapsulate a general SAX error or warning.
<u>SAXNotRecognizedException</u>	Exception class for an unrecognized identifier.
<u>SAXNotSupportedException</u>	Exception class for an unsupported operation.
<u>SAXParseException</u>	Encapsulate an XML parse error or warning.

[Overview](#) **Package** [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV PACKAGE](#) [NEXT PACKAGE](#)

[FRAMES](#) [NO FRAMES](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV PACKAGE](#) [NEXT PACKAGE](#)[FRAMES](#) [NO FRAMES](#)

Package org.xml.sax.ext

Interface Summary

<i><u>DeclHandler</u></i>	SAX2 extension handler for DTD declaration events.
<i><u>LexicalHandler</u></i>	SAX2 extension handler for lexical events.

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV PACKAGE](#) [NEXT PACKAGE](#)[FRAMES](#) [NO FRAMES](#)

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV PACKAGE](#)
[NEXT PACKAGE](#)
[FRAMES](#)
[NO FRAMES](#)

Package org.xml.sax.helpers

Class Summary

AttributeListImpl	Deprecated. <i>This class implements a deprecated interface, AttributeList; that interface has been replaced by Attributes, which is implemented in the AttributesImpl helper class.</i>
AttributesImpl	Default implementation of the Attributes interface.
DefaultHandler	Default base class for SAX2 event handlers.
LocatorImpl	Provide an optional convenience implementation of Locator.
NamespaceSupport	Encapsulate Namespace logic for use by SAX drivers.
ParserAdapter	Adapt a SAX1 Parser as a SAX2 XMLReader.
ParserFactory	Deprecated. <i>This class works with the deprecated Parser interface.</i>
XMLFilterImpl	Base class for deriving an XML filter.
XMLReaderAdapter	Adapt a SAX2 XMLReader as a SAX1 Parser.
XMLReaderFactory	Factory for creating an XML reader.

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV PACKAGE](#)
[NEXT PACKAGE](#)
[FRAMES](#)
[NO FRAMES](#)

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
[SUMMARY: INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
[DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.parsers

Class ParserConfigurationException

```

java.lang.Object
|
+-- java.lang.Throwable
    |
    +-- java.lang.Exception
        |
        +-- javax.xml.parsers.ParserConfigurationException
  
```

public class **ParserConfigurationException**

extends java.lang.Exception

Indicates a serious configuration error.

Since:

JAXP 1.0

Version:

1.0

See Also:

[Serialized Form](#)

Constructor Summary

[ParserConfigurationException](#)()

Create a new ParserConfigurationException with no detail message.

[ParserConfigurationException](#)(java.lang.String msg)

Create a new ParserConfigurationException with the String specified as an error message.

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail**ParserConfigurationException**

```
public ParserConfigurationException()
```

Create a new ParserConfigurationException with no detail message.

ParserConfigurationException

```
public ParserConfigurationException(java.lang.String msg)
```

Create a new ParserConfigurationException with the String specified as an error message.

Parameters:

msg - The error message for the exception.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV](#) [NEXT](#)[FRAMES](#) [NO FRAMES](#)

Serialized Form

Package javax.xml.parsers

Class [javax.xml.parsers.FactoryConfigurationError](#)
implements Serializable

Serialized Fields

exception

java.lang.Exception **exception**

Class [javax.xml.parsers.ParserConfigurationException](#)
implements Serializable

Package javax.xml.transform

Class
[javax.xml.transform.TransformerConfigurationException](#)
implements Serializable

Class [javax.xml.transform.TransformerException](#)
implements Serializable

Serialized Fields

containedException

java.lang.Throwable **containedException**

Field containedException specifies a wrapped exception. May be null.

locator

[SourceLocator](#) **locator**

Field locator specifies where the error occurred

Class

[javax.xml.transform.TransformerFactoryConfigurationError](#)
implements Serializable

Serialized Fields

exception

`java.lang.Exception` **exception**

Package `org.w3c.dom`

Class [org.w3c.dom.DOMException](#) implements
Serializable

Serialized Fields

code

short **code**

Package `org.xml.sax`

Class [org.xml.sax.SAXException](#) implements
Serializable

Serialized Fields

exception

`java.lang.Exception` **exception**

The embedded exception if tunnelling, or null.

Class [org.xml.sax.SAXNotRecognizedException](#) implements Serializable

Class [org.xml.sax.SAXNotSupportedException](#) implements Serializable

Class [org.xml.sax.SAXParseException](#) implements Serializable

Serialized Fields

columnNumber

int **columnNumber**

The column number, or -1.

See Also:

[SAXParseException.getColumnNumber\(\)](#)

lineNumber

int **lineNumber**

The line number, or -1.

See Also:

[SAXParseException.getLineNumber\(\)](#)

publicId

java.lang.String **publicId**

The public identifier, or null.

See Also:

[SAXParseException.getPublicId\(\)](#)

systemId

java.lang.String **systemId**

The system identifier, or null.

See Also:

[SAXParseException.getSystemId\(\)](#)

[Overview](#) Package Class Use [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV](#) [NEXT](#)

[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.parsers

Class FactoryConfigurationError

```

java.lang.Object
|
+-- java.lang.Throwable
    |
    +-- java.lang.Error
        |
        +-- javax.xml.parsers.FactoryConfigurationError
  
```

public class **FactoryConfigurationError**

extends java.lang.Error

Thrown when a problem with configuration with the Parser Factories exists. This error will typically be thrown when the class of a parser factory specified in the system properties cannot be found or instantiated.

Since:

JAXP 1.0

Version:

1.0

See Also:

[Serialized Form](#)

Constructor Summary

[FactoryConfigurationError](#)()

Create a new FactoryConfigurationError with no detail message.

[FactoryConfigurationError](#)(java.lang.Exception e)

Create a new FactoryConfigurationError with a given Exception base cause of the error.

[FactoryConfigurationError](#)(java.lang.Exception e,
java.lang.String msg)

Create a new FactoryConfigurationError with the given Exception base cause and detail message.

[**FactoryConfigurationError**](#)(java.lang.String msg)

Create a new FactoryConfigurationError with the String specified as an error message.

Method Summary

java.lang.Exception	getException () Return the actual exception (if any) that caused this exception to be raised.
java.lang.String	getMessage () Return the message (if any) for this error .

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

FactoryConfigurationError

```
public FactoryConfigurationError( )
```

Create a new FactoryConfigurationError with no detail mesage.

FactoryConfigurationError

```
public FactoryConfigurationError( java.lang.String msg)
```

Create a new FactoryConfigurationError with the String specified as an error message.

Parameters:

msg - The error message for the exception.

FactoryConfigurationError

```
public FactoryConfigurationError( java.lang.Exception e)
```

Create a new FactoryConfigurationError with a given Exception base cause of the error.

Parameters:

e - The exception to be encapsulated in a FactoryConfigurationError.

FactoryConfigurationError

```
public FactoryConfigurationError( java.lang.Exception e,
                                   java.lang.String msg)
```

Create a new FactoryConfigurationError with the given Exception base cause and detail message.

Parameters:

e - The exception to be encapsulated in a FactoryConfigurationError

msg - The detail message.

e - The exception to be wrapped in a FactoryConfigurationError

Method Detail

getMessage

```
public java.lang.String getMessage()
```

Return the message (if any) for this error . If there is no message for the exception and there is an encapsulated exception then the message of that exception, if it exists will be returned. Else the name of the encapsulated exception will be returned.

Returns:

The error message.

Overrides:

getMessage in class java.lang.Throwable

getException

```
public java.lang.Exception getException( )
```

Return the actual exception (if any) that caused this exception to be raised.

Returns:

The encapsulated exception, or null if there is none.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

All Classes

[*Attr*](#)

[*AttributeList*](#)

[*AttributeListImpl*](#)

[*Attributes*](#)

[*AttributesImpl*](#)

[*CDATASection*](#)

[*CharacterData*](#)

[*Comment*](#)

[*ContentHandler*](#)

[*DeclHandler*](#)

[*DefaultHandler*](#)

[*Document*](#)

[*DocumentBuilder*](#)

[*DocumentBuilderFactory*](#)

[*DocumentFragment*](#)

[*DocumentHandler*](#)

[*DocumentType*](#)

[*DOMException*](#)

[*DOMImplementation*](#)

[*DOMLocator*](#)

[*DOMResult*](#)

[*DOMSource*](#)

[*DTDHandler*](#)

[*Element*](#)

[*Entity*](#)

[*EntityReference*](#)

[*EntityResolver*](#)

[*ErrorHandler*](#)

[*ErrorListener*](#)

[*FactoryConfigurationError*](#)

[*HandlerBase*](#)

[*InputSource*](#)

[*LexicalHandler*](#)

[*Locator*](#)

[*LocatorImpl*](#)

[*NamedNodeMap*](#)

[*NamespaceSupport*](#)

[*Node*](#)

[*NodeList*](#)

[Notation](#)

[OutputKeys](#)

[Parser](#)

[ParserAdapter](#)

[ParserConfigurationException](#)

[ParserFactory](#)

[ProcessingInstruction](#)

[Result](#)

[SAXException](#)

[SAXNotRecognizedException](#)

[SAXNotSupportedException](#)

[SAXParseException](#)

[SAXParser](#)

[SAXParserFactory](#)

[SAXResult](#)

[SAXSource](#)

[SAXTransformerFactory](#)

[Source](#)

[SourceLocator](#)

[StreamResult](#)

[StreamSource](#)

[Templates](#)

[TemplatesHandler](#)

[Text](#)

[Transformer](#)

[TransformerConfigurationException](#)

[TransformerException](#)

[TransformerFactory](#)

[TransformerFactoryConfigurationError](#)

[TransformerHandler](#)

[URIResolver](#)

[XMLFilter](#)

[XMLFilterImpl](#)

[XMLReader](#)

[XMLReaderAdapter](#)

[XMLReaderFactory](#)

[javax.xml.parsers](#)

Classes

[DocumentBuilder](#)

[DocumentBuilderFactory](#)

[SAXParser](#)

[SAXParserFactory](#)

Exceptions

[ParserConfigurationException](#)

Errors

[FactoryConfigurationError](#)

[javax.xml.transform](#)

Interfaces

[*ErrorListener*](#)

[*Result*](#)

[*Source*](#)

[*SourceLocator*](#)

[*Templates*](#)

[*URIResolver*](#)

Classes

[OutputKeys](#)

[Transformer](#)

[TransformerFactory](#)

Exceptions

[TransformerConfigurationException](#)

[TransformerException](#)

Errors

[TransformerFactoryConfigurationError](#)

[javax.xml.transform.dom](#)

Interfaces

[*DOMLocator*](#)

Classes

[DOMResult](#)

[DOMSource](#)

[javax.xml.transform.sax](#)

Interfaces

[*TemplatesHandler*](#)

[*TransformerHandler*](#)

Classes

[SAXResult](#)

[SAXSource](#)

[SAXTransformerFactory](#)

[javax.xml.transform.stream](#)

Classes

[StreamResult](#)

[StreamSource](#)

org.w3c.dom

Interfaces

[*Attr*](#)

[*CDATASection*](#)

[*CharacterData*](#)

[*Comment*](#)

[*Document*](#)

[*DocumentFragment*](#)

[*DocumentType*](#)

[*DOMImplementation*](#)

[*Element*](#)

[*Entity*](#)

[*EntityReference*](#)

[*NamedNodeMap*](#)

[*Node*](#)

[*NodeList*](#)

[*Notation*](#)

[*ProcessingInstruction*](#)

[*Text*](#)

Exceptions

[*DOMException*](#)

[org.xml.sax](#)

Interfaces

[*AttributeList*](#)

[*Attributes*](#)

[*ContentHandler*](#)

[*DocumentHandler*](#)

[*DTDHandler*](#)

[*EntityResolver*](#)

[*ErrorHandler*](#)

[*Locator*](#)

[*Parser*](#)

[*XMLFilter*](#)

[*XMLReader*](#)

Classes

[*HandlerBase*](#)

[*InputSource*](#)

Exceptions

[*SAXException*](#)

[*SAXNotRecognizedException*](#)

[*SAXNotSupportedException*](#)

[*SAXParseException*](#)

[org.xml.sax.ext](#)

Interfaces

[*DeclHandler*](#)

[*LexicalHandler*](#)

[org.xml.sax.helpers](#)

Classes

[AttributeListImpl](#)

[AttributesImpl](#)

[DefaultHandler](#)

[LocatorImpl](#)

[NamespaceSupport](#)

[ParserAdapter](#)

[ParserFactory](#)

[XMLFilterImpl](#)

[XMLReaderAdapter](#)

[XMLReaderFactory](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface Attr

public abstract interface **Attr**extends [Node](#)

The `Attr` interface represents an attribute in an `Element` object. Typically the allowable values for the attribute are defined in a document type definition.

`Attr` objects inherit the `Node` interface, but since they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree. Thus, the `Node` attributes `parentNode`, `previousSibling`, and `nextSibling` have a null value for `Attr` objects. The DOM takes the view that attributes are properties of elements rather than having a separate identity from the elements they are associated with; this should make it more efficient to implement such features as default attributes associated with all elements of a given type. Furthermore, `Attr` nodes may not be immediate children of a `DocumentFragment`. However, they can be associated with `Element` nodes contained within a `DocumentFragment`. In short, users and implementors of the DOM need to be aware that `Attr` nodes have some things in common with other objects inheriting the `Node` interface, but they also are quite distinct.

The attribute's effective value is determined as follows: if this attribute has been explicitly assigned any value, that value is the attribute's effective value; otherwise, if there is a declaration for this attribute, and that declaration includes a default value, then that default value is the attribute's effective value; otherwise, the attribute does not exist on this element in the structure model until it has been explicitly added. Note that the `nodeValue` attribute on the `Attr` instance can also be used to retrieve the string version of the attribute's value(s).

In XML, where the value of an attribute can contain entity references, the child nodes of the `Attr` node may be either `Text` or `EntityReference` nodes (when these are in use; see the description of `EntityReference` for discussion). Because the DOM Core is not aware of attribute types, it treats all attribute values as simple strings, even if the DTD or schema declares them as having tokenized types.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#), [CDATA_SECTION_NODE](#), [COMMENT_NODE](#),
[DOCUMENT_FRAGMENT_NODE](#), [DOCUMENT_NODE](#), [DOCUMENT_TYPE_NODE](#),
[ELEMENT_NODE](#), [ENTITY_NODE](#), [ENTITY_REFERENCE_NODE](#), [NOTATION_NODE](#),
[PROCESSING_INSTRUCTION_NODE](#), [TEXT_NODE](#)

Method Summary

java.lang.String	getName () Returns the name of this attribute.
Element	getOwnerElement () The Element node this attribute is attached to or null if this attribute is not in use.
boolean	getSpecified () If this attribute was explicitly given a value in the original document, this is true; otherwise, it is false.
java.lang.String	getValue () On retrieval, the value of the attribute is returned as a string.
void	setValue (java.lang.String value)

Methods inherited from interface org.w3c.dom.[Node](#)

[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#), [getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#), [getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#), [getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#), [hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#), [replaceChild](#), [setNodeValue](#), [setPrefix](#)

Method Detail

getName

```
public java.lang.String getName()
```

Returns the name of this attribute.

getSpecified

```
public boolean getSpecified()
```

If this attribute was explicitly given a value in the original document, this is `true`; otherwise, it is `false`. Note that the implementation is in charge of this attribute, not the user. If the user changes the value of the attribute (even if it ends up having the same value as the default value) then the specified flag is automatically flipped to `true`. To re-specify the attribute as the default value from the DTD, the user must delete the attribute. The implementation will then make a new attribute available with `specified` set to `false` and the default value (if one exists).

In summary: If the attribute has an assigned value in the document then `specified` is `true`, and the value is the assigned value. If the attribute has no assigned value in the document and has a default value in the DTD, then `specified` is `false`, and the value is the default value in the DTD. If the attribute has no assigned value in the document and has a value of `#IMPLIED` in the DTD, then the attribute does not appear in the structure model of the document. If the `ownerElement` attribute is `null` (i.e. because it was just created or was set to `null` by the various removal and cloning operations) `specified` is `true`.

getValue

```
public java.lang.String getValue()
```

On retrieval, the value of the attribute is returned as a string. Character and general entity references are replaced with their values. See also the method `getAttribute` on the `Element` interface.

On setting, this creates a `Text` node with the unparsed contents of the string. I.e. any characters that an XML processor would recognize as markup are instead treated as literal text. See also the method `setAttribute` on the `Element` interface.

Throws:

[DOMException](#) - `NO_MODIFICATION_ALLOWED_ERR`: Raised when the node is readonly.

setValue

```
public void setValue(java.lang.String value)
    throws DOMException
```

getOwnerElement

```
public Element getOwnerElement()
```

The `Element` node this attribute is attached to or `null` if this attribute is not in use.

Since:

DOM Level 2

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: INNER | FIELD | CONSTR | [METHOD](#)

DETAIL: FIELD | CONSTR | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface CDATASection

public abstract interface **CDATASection**

extends [Text](#)

CDATA sections are used to escape blocks of text containing characters that would otherwise be regarded as markup. The only delimiter that is recognized in a CDATA section is the "]]>" string that ends the CDATA section. CDATA sections cannot be nested. Their primary purpose is for including material such as XML fragments, without needing to escape all the delimiters.

The DOMString attribute of the Text node holds the text that is contained by the CDATA section. Note that this may contain characters that need to be escaped outside of CDATA sections and that, depending on the character encoding ("charset") chosen for serialization, it may be impossible to write out some characters as part of a CDATA section.

The CDATASection interface inherits from the CharacterData interface through the Text interface. Adjacent CDATASection nodes are not merged by use of the normalize method of the Node interface. Because no markup is recognized within a CDATASection, character numeric references cannot be used as an escape mechanism when serializing. Therefore, action needs to be taken when serializing a CDATASection with a character encoding where some of the contained characters cannot be represented. Failure to do so would not produce well-formed XML. One potential solution in the serialization process is to end the CDATA section before the character, output the character using a character reference or entity reference, and open a new CDATA section for any further characters in the text node. Note, however, that some code conversion libraries at the time of writing do not return an error or exception when a character is missing from the encoding, making the task of ensuring that data is not corrupted on serialization more difficult.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#), [CDATA_SECTION_NODE](#), [COMMENT_NODE](#),
[DOCUMENT_FRAGMENT_NODE](#), [DOCUMENT_NODE](#), [DOCUMENT_TYPE_NODE](#),
[ELEMENT_NODE](#), [ENTITY_NODE](#), [ENTITY_REFERENCE_NODE](#), [NOTATION_NODE](#),
[PROCESSING_INSTRUCTION_NODE](#), [TEXT_NODE](#)

Methods inherited from interface org.w3c.dom.[Text](#)

[splitText](#)**Methods inherited from interface org.w3c.dom.[CharacterData](#)**[appendData](#), [deleteData](#), [getData](#), [getLength](#), [insertData](#), [replaceData](#), [setData](#), [substringData](#)**Methods inherited from interface org.w3c.dom.[Node](#)**[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#), [getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#), [getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#), [getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#), [hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#), [replaceChild](#), [setNodeValue](#), [setPrefix](#)[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface CharacterData

All Known Subinterfaces:

[CDATASection](#), [Comment](#), [Text](#)
public abstract interface **CharacterData**extends [Node](#)

The CharacterData interface extends Node with a set of attributes and methods for accessing character data in the DOM. For clarity this set is defined here rather than on each object that uses these attributes and methods. No DOM objects correspond directly to CharacterData, though Text and others do inherit the interface from it. All `offsets` in this interface start from 0.

As explained in the DOMString interface, text strings in the DOM are represented in UTF-16, i.e. as a sequence of 16-bit units. In the following, the term 16-bit units is used whenever necessary to indicate that indexing on CharacterData is done in 16-bit units.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#), [CDATA_SECTION_NODE](#), [COMMENT_NODE](#),
[DOCUMENT_FRAGMENT_NODE](#), [DOCUMENT_NODE](#), [DOCUMENT_TYPE_NODE](#),
[ELEMENT_NODE](#), [ENTITY_NODE](#), [ENTITY_REFERENCE_NODE](#), [NOTATION_NODE](#),
[PROCESSING_INSTRUCTION_NODE](#), [TEXT_NODE](#)

Method Summary

void	appendData (java.lang.String arg) Append the string to the end of the character data of the node.
void	deleteData (int offset, int count) Remove a range of 16-bit units from the node.
java.lang.String	getData () The character data of the node that implements this interface.

int	<code>getLength()</code> The number of 16-bit units that are available through data and the <code>substringData</code> method below.
void	<code>insertData(int offset, java.lang.String arg)</code> Insert a string at the specified 16-bit unit offset.
void	<code>replaceData(int offset, int count, java.lang.String arg)</code> Replace the characters starting at the specified 16-bit unit offset with the specified string.
void	<code>setData(java.lang.String data)</code>
java.lang.String	<code>substringData(int offset, int count)</code> Extracts a range of data from the node.

Methods inherited from interface [org.w3c.dom.Node](#)

[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#), [getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#), [getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#), [getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#), [hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#), [replaceChild](#), [setNodeValue](#), [setPrefix](#)

Method Detail

getData

```
public java.lang.String getData()
    throws DOMException
```

The character data of the node that implements this interface. The DOM implementation may not put arbitrary limits on the amount of data that may be stored in a `CharacterData` node. However, implementation limits may mean that the entirety of a node's data may not fit into a single `DOMString`. In such cases, the user may call `substringData` to retrieve the data in appropriately sized pieces.

Throws:

[DOMException](#) - `NO_MODIFICATION_ALLOWED_ERR`: Raised when the node is readonly.

[DOMException](#) - DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a DOMString variable on the implementation platform.

setData

```
public void setData(java.lang.String data)
    throws DOMException
```

getLength

```
public int getLength()
```

The number of 16-bit units that are available through data and the substringData method below. This may have the value zero, i.e., CharacterData nodes may be empty.

substringData

```
public java.lang.String substringData(int offset,
                                         int count)
    throws DOMException
```

Extracts a range of data from the node.

Parameters:

offsetStart - offset of substring to extract.

countThe - number of 16-bit units to extract.

Returns:

The specified substring. If the sum of offset and count exceeds the length, then all 16-bit units to the end of the data are returned.

Throws:

[DOMException](#) - INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.
DOMSTRING_SIZE_ERR: Raised if the specified range of text does not fit into a DOMString.

appendData

```
public void appendData(java.lang.String arg)
```

throws [DOMException](#)

Append the string to the end of the character data of the node. Upon success, data provides access to the concatenation of data and the DOMString specified.

Parameters:

argThe - DOMString to append.

Throws:

[DOMException](#) - NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

insertData

```
public void insertData(int offset,  
                        java.lang.String arg)  
    throws DOMException
```

Insert a string at the specified 16-bit unit offset.

Parameters:

offsetThe - character offset at which to insert.

argThe - DOMString to insert.

Throws:

[DOMException](#) - INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

deleteData

```
public void deleteData(int offset,  
                        int count)  
    throws DOMException
```

Remove a range of 16-bit units from the node. Upon success, data and length reflect the change.

Parameters:

offsetThe - offset from which to start removing.

countThe - number of 16-bit units to delete. If the sum of offset and count exceeds length then all 16-bit units from offset to the end of the data are deleted.

Throws:

[DOMException](#) - INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

replaceData

```
public void replaceData(int offset,  
                        int count,  
                        java.lang.String arg)  
    throws DOMException
```

Replace the characters starting at the specified 16-bit unit offset with the specified string.

Parameters:

`offset` - offset from which to start replacing.

`count` - number of 16-bit units to replace. If the sum of `offset` and `count` exceeds `length`, then all 16-bit units to the end of the data are replaced; (i.e., the effect is the same as a `remove` method call with the same range, followed by an `append` method invocation).

`arg` - `DOMString` with which the range must be replaced.

Throws:

[DOMException](#) - INDEX_SIZE_ERR: Raised if the specified `offset` is negative or greater than the number of 16-bit units in `data`, or if the specified `count` is negative.
NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)

SUMMARY: INNER | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

org.w3c.dom

Interface Comment

public abstract interface **Comment**extends [CharacterData](#)

This interface inherits from `CharacterData` and represents the content of a comment, i.e., all the characters between the starting '`<!--`' and ending '`-->`'. Note that this is the definition of a comment in XML, and, in practice, HTML, although some HTML tools may implement the full SGML comment structure.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#), [CDATA_SECTION_NODE](#), [COMMENT_NODE](#),
[DOCUMENT_FRAGMENT_NODE](#), [DOCUMENT_NODE](#), [DOCUMENT_TYPE_NODE](#),
[ELEMENT_NODE](#), [ENTITY_NODE](#), [ENTITY_REFERENCE_NODE](#), [NOTATION_NODE](#),
[PROCESSING_INSTRUCTION_NODE](#), [TEXT_NODE](#)

Methods inherited from interface org.w3c.dom.[CharacterData](#)

[appendData](#), [deleteData](#), [getData](#), [getLength](#), [insertData](#), [replaceData](#),
[setData](#), [substringData](#)

Methods inherited from interface org.w3c.dom.[Node](#)

[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#),
[getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#),
[getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#),
[getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#),
[hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#),
[replaceChild](#), [setNodeValue](#), [setPrefix](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: INNER | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)**org.xml.sax.ext**

Interface DeclHandler

public abstract interface **DeclHandler**

SAX2 extension handler for DTD declaration events.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This is an optional extension handler for SAX2 to provide information about DTD declarations in an XML document. XML readers are not required to support this handler, and this handler is not included in the core SAX2 distribution.

Note that data-related DTD declarations (unparsed entities and notations) are already reported through the [DTDHandler](#) interface.

If you are using the declaration handler together with a lexical handler, all of the events will occur between the [startDTD](#) and the [endDTD](#) events.

To set the DeclHandler for an XML reader, use the [setProperty](#) method with the propertyId "http://xml.org/sax/properties/declaration-handler". If the reader does not support declaration events, it will throw a [SAXNotRecognizedException](#) or a [SAXNotSupportedException](#) when you attempt to register the handler.

Since:

1.0

Version:

1.0

Author:

David Megginson, sax@megginson.com

See Also:

[XMLReader](#)

Method Summary

void	attributeDecl (java.lang.String eName, java.lang.String aName, java.lang.String type, java.lang.String valueDefault, java.lang.String value) Report an attribute type declaration.
void	elementDecl (java.lang.String name, java.lang.String model) Report an element type declaration.
void	externalEntityDecl (java.lang.String name, java.lang.String publicId, java.lang.String systemId) Report a parsed external entity declaration.
void	internalEntityDecl (java.lang.String name, java.lang.String value) Report an internal entity declaration.

Method Detail

elementDecl

```
public void elementDecl(java.lang.String name,
                        java.lang.String model)
    throws SAXException
```

Report an element type declaration.

The content model will consist of the string "EMPTY", the string "ANY", or a parenthesised group, optionally followed by an occurrence indicator. The model will be normalized so that all parameter entities are fully resolved and all whitespace is removed, and will include the enclosing parentheses. Other normalization (such as removing redundant parentheses or simplifying occurrence indicators) is at the discretion of the parser.

Parameters:

name - The element type name.

model - The content model as a normalized string.

Throws:

[SAXException](#) - The application may raise an exception.

attributeDecl

```
public void attributeDecl(java.lang.String eName,
                        java.lang.String aName,
```

```

        java.lang.String type,
        java.lang.String valueDefault,
        java.lang.String value)
    throws SAXException

```

Report an attribute type declaration.

Only the effective (first) declaration for an attribute will be reported. The type will be one of the strings "CDATA", "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", "ENTITIES", a parenthesized token group with the separator "|" and all whitespace removed, or the word "NOTATION" followed by a space followed by a parenthesized token group with all whitespace removed.

Any parameter entities in the attribute value will be expanded, but general entities will not.

Parameters:

eName - The name of the associated element.

aName - The name of the attribute.

type - A string representing the attribute type.

valueDefault - A string representing the attribute default ("#IMPLIED", "#REQUIRED", or "#FIXED") or null if none of these applies.

value - A string representing the attribute's default value, or null if there is none.

Throws:

[SAXException](#) - The application may raise an exception.

internalEntityDecl

```

public void internalEntityDecl(java.lang.String name,
                                java.lang.String value)
    throws SAXException

```

Report an internal entity declaration.

Only the effective (first) declaration for each entity will be reported. All parameter entities in the value will be expanded, but general entities will not.

Parameters:

name - The name of the entity. If it is a parameter entity, the name will begin with '%'.

value - The replacement text of the entity.

Throws:

[SAXException](#) - The application may raise an exception.

See Also:

[externalEntityDecl\(java.lang.String, java.lang.String, java.lang.String\)](#),

[DTDHandler.unparsedEntityDecl\(java.lang.String,
java.lang.String, java.lang.String, java.lang.String\)](#)

externalEntityDecl

```
public void externalEntityDecl(java.lang.String name,  
                                java.lang.String publicId,  
                                java.lang.String systemId)  
    throws SAXException
```

Report a parsed external entity declaration.

Only the effective (first) declaration for each entity will be reported.

Parameters:

name - The name of the entity. If it is a parameter entity, the name will begin with '%'.

publicId - The declared public identifier of the entity, or null if none was declared.

systemId - The declared system identifier of the entity.

Throws:

[SAXException](#) - The application may raise an exception.

See Also:

[internalEntityDecl\(java.lang.String, java.lang.String\)](#),
[DTDHandler.unparsedEntityDecl\(java.lang.String,
java.lang.String, java.lang.String, java.lang.String\)](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface Document

public abstract interface **Document**extends [Node](#)

The Document interface represents the entire HTML or XML document. Conceptually, it is the root of the document tree, and provides the primary access to the document's data.

Since elements, text nodes, comments, processing instructions, etc. cannot exist outside the context of a Document, the Document interface also contains the factory methods needed to create these objects. The Node objects created have a ownerDocument attribute which associates them with the Document within whose context they were created.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#), [CDATA_SECTION_NODE](#), [COMMENT_NODE](#),
[DOCUMENT_FRAGMENT_NODE](#), [DOCUMENT_NODE](#), [DOCUMENT_TYPE_NODE](#),
[ELEMENT_NODE](#), [ENTITY_NODE](#), [ENTITY_REFERENCE_NODE](#), [NOTATION_NODE](#),
[PROCESSING_INSTRUCTION_NODE](#), [TEXT_NODE](#)

Method Summary

Attr	createAttribute (java.lang.String name) Creates an Attr of the given name.
Attr	createAttributeNS (java.lang.String namespaceURI, java.lang.String qualifiedName) Creates an attribute of the given qualified name and namespace URI.
CDATASection	createCDATASection (java.lang.String data) Creates a CDATASection node whose value is the specified string.
Comment	createComment (java.lang.String data) Creates a Comment node given the specified string.
DocumentFragment	createDocumentFragment () Creates an empty DocumentFragment object.
Element	createElement (java.lang.String tagName) Creates an element of the type specified.
Element	createElementNS (java.lang.String namespaceURI, java.lang.String qualifiedName) Creates an element of the given qualified name and namespace URI.

EntityReference	createEntityReference (java.lang.String name) Creates an EntityReference object.
ProcessingInstruction	createProcessingInstruction (java.lang.String target, java.lang.String data) Creates a ProcessingInstruction node given the specified name and data strings.
Text	createTextNode (java.lang.String data) Creates a Text node given the specified string.
DocumentType	getDoctype () The Document Type Declaration (see DocumentType) associated with this document.
Element	getDocumentElement () This is a convenience attribute that allows direct access to the child node that is the root element of the document.
Element	getElementById (java.lang.String elementId) Returns the Element whose ID is given by elementId.
NodeList	getElementsByTagName (java.lang.String tagname) Returns a NodeList of all the Elements with a given tag name in the order in which they are encountered in a preorder traversal of the Document tree.
NodeList	getElementsByTagNameNS (java.lang.String namespaceURI, java.lang.String localName) Returns a NodeList of all the Elements with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of the Document tree.
DOMImplementation	getImplementation () The DOMImplementation object that handles this document.
Node	importNode (Node importedNode, boolean deep) Imports a node from another document to this document.

Methods inherited from interface org.w3c.dom.[Node](#)

[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#), [getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#), [getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#), [getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#), [hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#), [replaceChild](#), [setNodeValue](#), [setPrefix](#)

Method Detail

getDoctype

```
public DocumentType getDoctype()
```

The Document Type Declaration (see [DocumentType](#)) associated with this document. For HTML documents as well as XML documents without a document type declaration this returns `null`. The DOM Level 2 does not support editing the Document Type Declaration. `doctype` cannot be altered in any way, including through the use of methods inherited from the [Node](#) interface, such as `insertNode` or `removeNode`.

getImplementation

```
public DOMImplementation getImplementation()
```

The [DOMImplementation](#) object that handles this document. A DOM application may use objects from multiple implementations.

getDocumentElement

```
public Element getDocumentElement()
```

This is a convenience attribute that allows direct access to the child node that is the root element of the document. For HTML documents, this is the element with the `tagName` "HTML".

createElement

```
public Element createElement(java.lang.String tagName)
    throws DOMException
```

Creates an element of the type specified. Note that the instance returned implements the [Element](#) interface, so attributes can be specified directly on the returned object.

In addition, if there are known attributes with default values, `Attr` nodes representing them are automatically created and attached to the element.

To create an element with a qualified name and namespace URI, use the `createElementNS` method.

Parameters:

`tagName`The - name of the element type to instantiate. For XML, this is case-sensitive. For HTML, the `tagName` parameter may be provided in any case, but it must be mapped to the canonical uppercase form by the DOM implementation.

Returns:

A new [Element](#) object with the `nodeName` attribute set to `tagName`, and `localName`, `prefix`, and `namespaceURI` set to `null`.

Throws:

[DOMException](#) - `INVALID_CHARACTER_ERR`: Raised if the specified name contains an illegal character.

createDocumentFragment

```
public DocumentFragment createDocumentFragment()
```

Creates an empty DocumentFragment object.

Returns:

A new DocumentFragment.

createTextNode

```
public Text createTextNode(java.lang.String data)
```

Creates a Text node given the specified string.

Parameters:

dataThe - data for the node.

Returns:

The new Text object.

createComment

```
public Comment createComment(java.lang.String data)
```

Creates a Comment node given the specified string.

Parameters:

dataThe - data for the node.

Returns:

The new Comment object.

createCDATASection

```
public CDATASection createCDATASection(java.lang.String data)
                                   throws DOMException
```

Creates a CDATASection node whose value is the specified string.

Parameters:

dataThe - data for the CDATASection contents.

Returns:

The new CDATASection object.

Throws:

[DOMException](#) - NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

createProcessingInstruction

```
public ProcessingInstruction createProcessingInstruction(java.lang.String target,
                                                         java.lang.String data)
                                                         throws DOMException
```

Creates a ProcessingInstruction node given the specified name and data strings.

Parameters:

targetThe - target part of the processing instruction.

dataThe - data for the node.

Returns:

The new ProcessingInstruction object.

Throws:

[DOMException](#) - INVALID_CHARACTER_ERR: Raised if the specified target contains an illegal character.

NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

createAttribute

```
public Attr createAttribute(java.lang.String name)
                           throws DOMException
```

Creates an Attr of the given name. Note that the Attr instance can then be set on an Element using the setAttributeNode method.

To create an attribute with a qualified name and namespace URI, use the createAttributeNS method.

Parameters:

nameThe - name of the attribute.

Returns:

A new Attr object with the nodeName attribute set to name, and localName, prefix, and namespaceURI set to null. The value of the attribute is the empty string.

Throws:

[DOMException](#) - INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

createEntityReference

```
public EntityReference createEntityReference(java.lang.String name)
                                             throws DOMException
```

Creates an EntityReference object. In addition, if the referenced entity is known, the child list of the EntityReference node is made the same as that of the corresponding Entity node. If any descendant of the Entity node has an unbound namespace prefix, the corresponding descendant of the created EntityReference node is also unbound; (its namespaceURI is null). The DOM Level 2 does not support any mechanism to resolve namespace prefixes.

Parameters:

nameThe - name of the entity to reference.

Returns:

The new `EntityReference` object.

Throws:

[DOMException](#) - `INVALID_CHARACTER_ERR`: Raised if the specified name contains an illegal character.

`NOT_SUPPORTED_ERR`: Raised if this document is an HTML document.

getElementsByTagName

```
public NodeList getElementsByTagName(java.lang.String tagname)
```

Returns a `NodeList` of all the `Elements` with a given tag name in the order in which they are encountered in a preorder traversal of the `Document` tree.

Parameters:

tagnameThe - name of the tag to match on. The special value "*" matches all tags.

Returns:

A new `NodeList` object containing all the matched `Elements`.

importNode

```
public Node importNode(Node importedNode,  
                        boolean deep)  
    throws DOMException
```

Imports a node from another document to this document. The returned node has no parent; (`parentNode` is `null`). The source node is not altered or removed from the original document; this method creates a new copy of the source node.

For all nodes, importing a node creates a node object owned by the importing document, with attribute values identical to the source node's `nodeName` and `nodeType`, plus the attributes related to namespaces (`prefix`, `localName`, and `namespaceURI`). As in the `cloneNode` operation on a `Node`, the source node is not altered. Additional information is copied as appropriate to the `nodeType`, attempting to mirror the behavior expected if a fragment of XML or HTML source was copied from one document to another, recognizing that the two documents may have different DTDs in the XML case. The following list describes the specifics for each type of node.

ATTRIBUTE_NODE

The `ownerElement` attribute is set to `null` and the specified flag is set to `true` on the generated `Attr`. The descendants of the source `Attr` are recursively imported and the resulting nodes reassembled to form the corresponding subtree. Note that the `deep` parameter has no effect on `Attr` nodes; they always carry their children with them when imported.

DOCUMENT_FRAGMENT_NODE

If the `deep` option was set to `true`, the descendants of the source element are recursively imported and the resulting nodes reassembled to form the corresponding subtree. Otherwise, this simply generates an empty `DocumentFragment`.

DOCUMENT_NODE

Document nodes cannot be imported.

DOCUMENT_TYPE_NODE

DocumentType nodes cannot be imported.

ELEMENT_NODE

Specified attribute nodes of the source element are imported, and the generated Attr nodes are attached to the generated Element. Default attributes are not copied, though if the document being imported into defines default attributes for this element name, those are assigned. If the importNode deep parameter was set to true, the descendants of the source element are recursively imported and the resulting nodes reassembled to form the corresponding subtree.

ENTITY_NODE

Entity nodes can be imported, however in the current release of the DOM the DocumentType is readonly. Ability to add these imported nodes to a DocumentType will be considered for addition to a future release of the DOM. On import, the publicId, systemId, and notationName attributes are copied. If a deep import is requested, the descendants of the the source Entity are recursively imported and the resulting nodes reassembled to form the corresponding subtree.

ENTITY_REFERENCE_NODE

Only the EntityReference itself is copied, even if a deep import is requested, since the source and destination documents might have defined the entity differently. If the document being imported into provides a definition for this entity name, its value is assigned.

NOTATION_NODE

Notation nodes can be imported, however in the current release of the DOM the DocumentType is readonly. Ability to add these imported nodes to a DocumentType will be considered for addition to a future release of the DOM. On import, the publicId and systemId attributes are copied. Note that the deep parameter has no effect on Notation nodes since they never have any children.

PROCESSING_INSTRUCTION_NODE

The imported node copies its target and data values from those of the source node.

TEXT_NODE, CDATA_SECTION_NODE, COMMENT_NODE

These three types of nodes inheriting from CharacterData copy their data and length attributes from those of the source node.

Parameters:

importedNodeThe - node to import.

deepIf - true, recursively import the subtree under the specified node; if false, import only the node itself, as explained above. This has no effect on Attr, EntityReference, and Notation nodes.

Returns:

The imported node that belongs to this Document.

Throws:

[DOMException](#) - NOT_SUPPORTED_ERR: Raised if the type of node being imported is not supported.

Since:

DOM Level 2

createElementNS

```
public Element createElementNS( java.lang.String namespaceURI,
                                java.lang.String qualifiedName)
```

throws [DOMException](#)

Creates an element of the given qualified name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters:

namespaceURIThe - namespace URI of the element to create.

qualifiedNameThe - qualified name of the element type to instantiate.

Returns:

A new `Element` object with the following attributes: `AttributeValueNode.nodeName`
`qualifiedNameNode.namespaceURI` `namespaceURINode.prefix`prefix, extracted from
`qualifiedName`, or null if there is no `prefixNode.localName`local name, extracted from
`qualifiedNameElement.tagName` `qualifiedName`

Throws:

[DOMException](#) - `INVALID_CHARACTER_ERR`: Raised if the specified qualified name contains an illegal character.

`NAMESPACE_ERR`: Raised if the `qualifiedName` is malformed, if the `qualifiedName` has a prefix and the `namespaceURI` is null, or if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from " `http://www.w3.org/XML/1998/namespace`".

Since:

DOM Level 2

createAttributeNS

```
public Attr createAttributeNS(java.lang.String namespaceURI,
                               java.lang.String qualifiedName)
                               throws DOMException
```

Creates an attribute of the given qualified name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters:

namespaceURIThe - namespace URI of the attribute to create.

qualifiedNameThe - qualified name of the attribute to instantiate.

Returns:

A new `Attr` object with the following attributes: `AttributeValueNode.nodeName`
`qualifiedNameNode.namespaceURI` `namespaceURI` `Node.prefix`prefix, extracted from
`qualifiedName`, or null if there is no `prefixNode.localName`local name, extracted from
`qualifiedNameAttr.name` `qualifiedNameNode.nodeValue`the empty string

Throws:

[DOMException](#) - `INVALID_CHARACTER_ERR`: Raised if the specified qualified name contains an illegal character.

`NAMESPACE_ERR`: Raised if the `qualifiedName` is malformed, if the `qualifiedName` has a prefix and the `namespaceURI` is null, if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from " `http://www.w3.org/XML/1998/namespace`", or if the `qualifiedName` is "xmlns" and the `namespaceURI` is different from " `http://www.w3.org/2000/xmlns/`".

Since:

DOM Level 2

getElementsByTagNameNS

```
public NodeList getElementsByTagNameNS(java.lang.String namespaceURI,  
                                           java.lang.String localName)
```

Returns a `NodeList` of all the `Elements` with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of the `Document` tree.

Parameters:

`namespaceURI`The - namespace URI of the elements to match on. The special value "*" matches all namespaces.

`localName`The - local name of the elements to match on. The special value "*" matches all local names.

Returns:

A new `NodeList` object containing all the matched `Elements`.

Since:

DOM Level 2

getElementById

```
public Element getElementById(java.lang.String elementId)
```

Returns the `Element` whose ID is given by `elementId`. If no such element exists, returns `null`. Behavior is not defined if more than one element has this ID. The DOM implementation must have information that says which attributes are of type ID. Attributes with the name "ID" are not of type ID unless so defined. Implementations that do not know whether attributes are of type ID or not are expected to return `null`.

Parameters:

`elementId`The - unique id value for an element.

Returns:

The matching element.

Since:

DOM Level 2

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface DocumentFragment

public abstract interface **DocumentFragment**

extends [Node](#)

DocumentFragment is a "lightweight" or "minimal" Document object. It is very common to want to be able to extract a portion of a document's tree or to create a new fragment of a document. Imagine implementing a user command like cut or rearranging a document by moving fragments around. It is desirable to have an object which can hold such fragments and it is quite natural to use a Node for this purpose. While it is true that a Document object could fulfill this role, a Document object can potentially be a heavyweight object, depending on the underlying implementation. What is really needed for this is a very lightweight object. DocumentFragment is such an object.

Furthermore, various operations -- such as inserting nodes as children of another Node -- may take DocumentFragment objects as arguments; this results in all the child nodes of the DocumentFragment being moved to the child list of this node.

The children of a DocumentFragment node are zero or more nodes representing the tops of any sub-trees defining the structure of the document. DocumentFragment nodes do not need to be well-formed XML documents (although they do need to follow the rules imposed upon well-formed XML parsed entities, which can have multiple top nodes). For example, a DocumentFragment might have only one child and that child node could be a Text node. Such a structure model represents neither an HTML document nor a well-formed XML document.

When a DocumentFragment is inserted into a Document (or indeed any other Node that may take children) the children of the DocumentFragment and not the DocumentFragment itself are inserted into the Node. This makes the DocumentFragment very useful when the user wishes to create nodes that are siblings; the DocumentFragment acts as the parent of these nodes so that the user can use the standard methods from the Node interface, such as `insertBefore` and `appendChild`.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#), [CDATA_SECTION_NODE](#), [COMMENT_NODE](#),
[DOCUMENT_FRAGMENT_NODE](#), [DOCUMENT_NODE](#), [DOCUMENT_TYPE_NODE](#),
[ELEMENT_NODE](#), [ENTITY_NODE](#), [ENTITY_REFERENCE_NODE](#), [NOTATION_NODE](#),
[PROCESSING_INSTRUCTION_NODE](#), [TEXT_NODE](#)

Methods inherited from interface [org.w3c.dom.Node](#)

[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#), [getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#), [getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#), [getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#), [hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#), [replaceChild](#), [setNodeValue](#), [setPrefix](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface DocumentType

public abstract interface **DocumentType**

extends [Node](#)

Each Document has a doctype attribute whose value is either null or a DocumentType object. The DocumentType interface in the DOM Core provides an interface to the list of entities that are defined for the document, and little else because the effect of namespaces and the various XML schema efforts on DTD representation are not clearly understood as of this writing.

The DOM Level 2 doesn't support editing DocumentType nodes.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#), [CDATA_SECTION_NODE](#), [COMMENT_NODE](#),
[DOCUMENT_FRAGMENT_NODE](#), [DOCUMENT_NODE](#), [DOCUMENT_TYPE_NODE](#),
[ELEMENT_NODE](#), [ENTITY_NODE](#), [ENTITY_REFERENCE_NODE](#), [NOTATION_NODE](#),
[PROCESSING_INSTRUCTION_NODE](#), [TEXT_NODE](#)

Method Summary

NamedNodeMap	getEntities () A NamedNodeMap containing the general entities, both external and internal, declared in the DTD.
java.lang.String	getInternalSubset () The internal subset as a string. The actual content returned depends on how much information is available to the implementation.
java.lang.String	getName () The name of DTD; i.e., the name immediately following the DOCTYPE keyword.
NamedNodeMap	getNotations () A NamedNodeMap containing the notations declared in the DTD.

java.lang.String	<code>getPublicId()</code> The public identifier of the external subset.
java.lang.String	<code>getSystemId()</code> The system identifier of the external subset.

Methods inherited from interface org.w3c.dom.[Node](#)

[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#), [getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#), [getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#), [getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#), [hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#), [replaceChild](#), [setNodeValue](#), [setPrefix](#)

Method Detail

getName

```
public java.lang.String getName()
```

The name of DTD; i.e., the name immediately following the DOCTYPE keyword.

getEntities

```
public NamedNodeMap getEntities()
```

A [NamedNodeMap](#) containing the general entities, both external and internal, declared in the DTD. Parameter entities are not contained. Duplicates are discarded. For example in:

```
<!DOCTYPE
  ex SYSTEM "ex.dtd" [ <!ENTITY foo "foo"> <!ENTITY bar
    "bar"> <!ENTITY bar "bar2"> <!ENTITY % baz "baz">
  ]> <ex/>
```

the interface provides access to `foo` and the first declaration of `bar` but not the second declaration of `bar` or `baz`. Every node in this map also implements the [Entity](#) interface.

The DOM Level 2 does not support editing entities, therefore `entities` cannot be altered in any way.

getNotations

```
public NamedNodeMap getNotations()
```

A `NamedNodeMap` containing the notations declared in the DTD. Duplicates are discarded. Every node in this map also implements the `Notation` interface.

The DOM Level 2 does not support editing notations, therefore `notations` cannot be altered in any way.

getPublicId

```
public java.lang.String getPublicId()
```

The public identifier of the external subset.

Since:

DOM Level 2

getSystemId

```
public java.lang.String getSystemId()
```

The system identifier of the external subset.

Since:

DOM Level 2

getInternalSubset

```
public java.lang.String getInternalSubset()
```

The internal subset as a string. The actual content returned depends on how much information is available to the implementation. This may vary depending on various parameters, including the XML processor used to build the document.

Since:

DOM Level 2

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#)
[Package](#)
[Class](#)
[Use Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
[FRAMES](#)
[NO FRAMES](#)
[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Class DOMException

```

java.lang.Object
|
+-- java.lang.Throwable
    |
    +-- java.lang.Exception
        |
        +-- java.lang.RuntimeException
            |
            +-- org.w3c.dom.DOMException
  
```

public class **DOMException**

extends java.lang.RuntimeException

DOM operations only raise exceptions in "exceptional" circumstances, i.e., when an operation is impossible to perform (either for logical reasons, because data is lost, or because the implementation has become unstable). In general, DOM methods return specific error values in ordinary processing situations, such as out-of-bound errors when using `NodeList`.

Implementations should raise other exceptions under other circumstances. For example, implementations should raise an implementation-dependent exception if a `null` argument is passed.

Some languages and object systems do not support the concept of exceptions. For such systems, error conditions may be indicated using native error reporting mechanisms. For some bindings, for example, methods may return error codes similar to those listed in the corresponding method descriptions.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

See Also:

[Serialized Form](#)

Field Summary

short	code
static short	DOMSTRING SIZE ERR If the specified range of text does not fit into a DOMString

static short	<u>HIERARCHY_REQUEST_ERR</u> If any node is inserted somewhere it doesn't belong
static short	<u>INDEX_SIZE_ERR</u> If index or size is negative, or greater than the allowed value
static short	<u>INUSE_ATTRIBUTE_ERR</u> If an attempt is made to add an attribute that is already in use elsewhere
static short	<u>INVALID_ACCESS_ERR</u> If a parameter or an operation is not supported by the underlying object.
static short	<u>INVALID_CHARACTER_ERR</u> If an invalid or illegal character is specified, such as in a name.
static short	<u>INVALID_MODIFICATION_ERR</u> If an attempt is made to modify the type of the underlying object.
static short	<u>INVALID_STATE_ERR</u> If an attempt is made to use an object that is not, or is no longer, usable.
static short	<u>NAMESPACE_ERR</u> If an attempt is made to create or change an object in a way which is incorrect with regard to namespaces.
static short	<u>NO_DATA_ALLOWED_ERR</u> If data is specified for a node which does not support data
static short	<u>NO_MODIFICATION_ALLOWED_ERR</u> If an attempt is made to modify an object where modifications are not allowed
static short	<u>NOT_FOUND_ERR</u> If an attempt is made to reference a node in a context where it does not exist
static short	<u>NOT_SUPPORTED_ERR</u> If the implementation does not support the requested type of object or operation.
static short	<u>SYNTAX_ERR</u> If an invalid or illegal string is specified.
static short	<u>WRONG_DOCUMENT_ERR</u> If a node is used in a different document than the one that created it (that doesn't support it)

Constructor Summary

[DOMException](#)(short code, java.lang.String message)

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail**code**

```
public short code
```

INDEX_SIZE_ERR

```
public static final short INDEX_SIZE_ERR
```

If index or size is negative, or greater than the allowed value

DOMSTRING_SIZE_ERR

```
public static final short DOMSTRING_SIZE_ERR
```

If the specified range of text does not fit into a DOMString

HIERARCHY_REQUEST_ERR

```
public static final short HIERARCHY_REQUEST_ERR
```

If any node is inserted somewhere it doesn't belong

WRONG_DOCUMENT_ERR

```
public static final short WRONG_DOCUMENT_ERR
```

If a node is used in a different document than the one that created it (that doesn't support it)

INVALID_CHARACTER_ERR

```
public static final short INVALID_CHARACTER_ERR
```

If an invalid or illegal character is specified, such as in a name. See production 2 in the XML specification for the definition of a legal character, and production 5 for the definition of a legal name character.

NO_DATA_ALLOWED_ERR

```
public static final short NO_DATA_ALLOWED_ERR
```

If data is specified for a node which does not support data

NO_MODIFICATION_ALLOWED_ERR

```
public static final short NO_MODIFICATION_ALLOWED_ERR
```

If an attempt is made to modify an object where modifications are not allowed

NOT_FOUND_ERR

```
public static final short NOT_FOUND_ERR
```

If an attempt is made to reference a node in a context where it does not exist

NOT_SUPPORTED_ERR

```
public static final short NOT_SUPPORTED_ERR
```

If the implementation does not support the requested type of object or operation.

INUSE_ATTRIBUTE_ERR

`public static final short INUSE_ATTRIBUTE_ERR`

If an attempt is made to add an attribute that is already in use elsewhere

INVALID_STATE_ERR

`public static final short INVALID_STATE_ERR`

If an attempt is made to use an object that is not, or is no longer, usable.

Since:

DOM Level 2

SYNTAX_ERR

`public static final short SYNTAX_ERR`

If an invalid or illegal string is specified.

Since:

DOM Level 2

INVALID_MODIFICATION_ERR

`public static final short INVALID_MODIFICATION_ERR`

If an attempt is made to modify the type of the underlying object.

Since:

DOM Level 2

NAMESPACE_ERR

`public static final short NAMESPACE_ERR`

If an attempt is made to create or change an object in a way which is incorrect with regard to namespaces.

Since:

DOM Level 2

INVALID_ACCESS_ERR

```
public static final short INVALID_ACCESS_ERR
```

If a parameter or an operation is not supported by the underlying object.

Since:

DOM Level 2

Constructor Detail

DOMException

```
public DOMException(short code,  
                    java.lang.String message)
```

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface DOMImplementation

public abstract interface **DOMImplementation**

The DOMImplementation interface provides a number of methods for performing operations that are independent of any particular instance of the document object model.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Method Summary

Document	createDocument (java.lang.String namespaceURI , java.lang.String qualifiedName, DocumentType doctype) Creates an XML Document object of the specified type with its document element.
DocumentType	createDocumentType (java.lang.String qualifiedName , java.lang.String publicId, java.lang.String systemId) Creates an empty DocumentType node.
boolean	hasFeature (java.lang.String feature , java.lang.String version) Test if the DOM implementation implements a specific feature.

Method Detail

hasFeature

```
public boolean hasFeature(java.lang.String feature,
                           java.lang.String version)
```

Test if the DOM implementation implements a specific feature.

Parameters:

featureThe - name of the feature to test (case-insensitive). The values used by DOM features are defined throughout the DOM Level 2 specifications and listed in the section. The name must be an XML name. To avoid possible conflicts, as a convention, names

referring to features defined outside the DOM specification should be made unique by reversing the name of the Internet domain name of the person (or the organization that the person belongs to) who defines the feature, component by component, and using this as a prefix. For instance, the W3C SVG Working Group defines the feature "org.w3c.dom.svg".

`versionThis` - is the version number of the feature to test. In Level 2, the string can be either "2.0" or "1.0". If the version is not specified, supporting any version of the feature causes the method to return `true`.

Returns:

`true` if the feature is implemented in the specified version, `false` otherwise.

createDocumentType

```
public DocumentType createDocumentType( java.lang.String qualifiedName,
                                         java.lang.String publicId,
                                         java.lang.String systemId)
                                         throws DOMException
```

Creates an empty `DocumentType` node. Entity declarations and notations are not made available. Entity reference expansions and default attribute additions do not occur. It is expected that a future version of the DOM will provide a way for populating a `DocumentType`. HTML-only DOM implementations do not need to implement this method.

Parameters:

`qualifiedNameThe` - qualified name of the document type to be created.
`publicIdThe` - external subset public identifier.
`systemIdThe` - external subset system identifier.

Returns:

A new `DocumentType` node with `Node.ownerDocument` set to `null`.

Throws:

[DOMException](#) - `INVALID_CHARACTER_ERR`: Raised if the specified qualified name contains an illegal character.
`NAMESPACE_ERR`: Raised if the `qualifiedName` is malformed.

Since:

DOM Level 2

createDocument

```
public Document createDocument( java.lang.String namespaceURI,
                                 java.lang.String qualifiedName,
                                 DocumentType doctype)
```

throws [DOMException](#)

Creates an XML Document object of the specified type with its document element. HTML-only DOM implementations do not need to implement this method.

Parameters:

`namespaceURI`The - namespace URI of the document element to create.

`qualifiedName`The - qualified name of the document element to be created.

`doctype`The - type of document to be created or null. When doctype is not null, its `Node.ownerDocument` attribute is set to the document being created.

Returns:

A new Document object.

Throws:

[DOMException](#) - `INVALID_CHARACTER_ERR`: Raised if the specified qualified name contains an illegal character.

`NAMESPACE_ERR`: Raised if the `qualifiedName` is malformed, if the `qualifiedName` has a prefix and the `namespaceURI` is null, or if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from "http://www.w3.org/XML/1998/namespace".

`WRONG_DOCUMENT_ERR`: Raised if `doctype` has already been used with a different document or was created from a different implementation.

Since:

DOM Level 2

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
[FRAMES](#)
[NO FRAMES](#)
[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform.dom

Interface DOMLocator

public abstract interface **DOMLocator**extends [SourceLocator](#)

Indicates the position of a node in a source DOM, intended primarily for error reporting. To use a DOMLocator, the receiver of an error must downcast the [SourceLocator](#) object returned by an exception. A [Transformer](#) may use this object for purposes other than error reporting, for instance, to indicate the source node that originated a result node.

Method Summary

Node	getOriginatingNode ()
----------------------	--

Return the node where the event occurred.

Methods inherited from interface javax.xml.transform.[SourceLocator](#)

[getColumnNumber](#), [getLineNumber](#), [getPublicId](#), [getSystemId](#)

Method Detail

getOriginatingNode

public [Node](#) **getOriginatingNode** ()

Return the node where the event occurred.

Returns:

The node that is the location for the event.

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
[FRAMES](#)
[NO FRAMES](#)
[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface Element

public abstract interface **Element**extends [Node](#)

The `Element` interface represents an element in an HTML or XML document. Elements may have attributes associated with them; since the `Element` interface inherits from `Node`, the generic `Node` interface attribute `attributes` may be used to retrieve the set of all attributes for an element. There are methods on the `Element` interface to retrieve either an `Attr` object by name or an attribute value by name. In XML, where an attribute value may contain entity references, an `Attr` object should be retrieved to examine the possibly fairly complex sub-tree representing the attribute value. On the other hand, in HTML, where all attributes have simple string values, methods to directly access an attribute value can safely be used as a convenience. In DOM Level 2, the method `normalize` is inherited from the `Node` interface where it was moved.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#), [CDATA_SECTION_NODE](#), [COMMENT_NODE](#),
[DOCUMENT_FRAGMENT_NODE](#), [DOCUMENT_NODE](#), [DOCUMENT_TYPE_NODE](#),
[ELEMENT_NODE](#), [ENTITY_NODE](#), [ENTITY_REFERENCE_NODE](#), [NOTATION_NODE](#),
[PROCESSING_INSTRUCTION_NODE](#), [TEXT_NODE](#)

Method Summary

<code>java.lang.String</code>	getAttribute (<code>java.lang.String name</code>) Retrieves an attribute value by name.
Attr	getAttributeNode (<code>java.lang.String name</code>) Retrieves an attribute node by name.
Attr	getAttributeNodeNS (<code>java.lang.String namespaceURI</code> , <code>java.lang.String localName</code>) Retrieves an <code>Attr</code> node by local name and namespace URI.

java.lang.String	<u>getAttributeNS</u> (java.lang.String namespaceURI , java.lang.String localName) Retrieves an attribute value by local name and namespace URI.
<u>NodeList</u>	<u>getElementsByTagName</u> (java.lang.String name) Returns a <u>NodeList</u> of all descendant <u>Elements</u> with a given tag name, in the order in which they are encountered in a preorder traversal of this <u>Element</u> tree.
<u>NodeList</u>	<u>getElementsByTagNameNS</u> (java.lang.String namespaceURI , java.lang.String localName) Returns a <u>NodeList</u> of all the descendant <u>Elements</u> with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of this <u>Element</u> tree.
java.lang.String	<u>getTagName</u> () The name of the element.
boolean	<u>hasAttribute</u> (java.lang.String name) Returns true when an attribute with a given name is specified on this element or has a default value, false otherwise.
boolean	<u>hasAttributeNS</u> (java.lang.String namespaceURI , java.lang.String localName) Returns true when an attribute with a given local name and namespace URI is specified on this element or has a default value, false otherwise.
void	<u>removeAttribute</u> (java.lang.String name) Removes an attribute by name.
<u>Attr</u>	<u>removeAttributeNode</u> (<u>Attr</u> oldAttr) Removes the specified attribute node.
void	<u>removeAttributeNS</u> (java.lang.String namespaceURI , java.lang.String localName) Removes an attribute by local name and namespace URI.
void	<u>setAttribute</u> (java.lang.String name , java.lang.String value) Adds a new attribute.
<u>Attr</u>	<u>setAttributeNode</u> (<u>Attr</u> newAttr) Adds a new attribute node.
<u>Attr</u>	<u>setAttributeNodeNS</u> (<u>Attr</u> newAttr) Adds a new attribute.
void	<u>setAttributeNS</u> (java.lang.String namespaceURI , java.lang.String qualifiedName , java.lang.String value) Adds a new attribute.

Methods inherited from interface [org.w3c.dom.Node](#)

[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#), [getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#), [getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#), [getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#), [hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#), [replaceChild](#), [setNodeValue](#), [setPrefix](#)

Method Detail

getTagName

```
public java.lang.String getTagName()
```

The name of the element. For example, in:

```
<elementExample
id="demo"> ... </elementExample> ,
```

tagName has the value "elementExample". Note that this is case-preserving in XML, as are all of the operations of the DOM. The HTML DOM returns the tagName of an HTML element in the canonical uppercase form, regardless of the case in the source HTML document.

getAttribute

```
public java.lang.String getAttribute(java.lang.String name)
```

Retrieves an attribute value by name.

Parameters:

nameThe - name of the attribute to retrieve.

Returns:

The Attr value as a string, or the empty string if that attribute does not have a specified or default value.

setAttribute

```
public void setAttribute(java.lang.String name,
                           java.lang.String value)
```

throws [DOMException](#)

Adds a new attribute. If an attribute with that name is already present in the element, its value is changed to be that of the value parameter. This value is a simple string; it is not parsed as it is being set. So any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately escaped by the implementation when it is written out. In order to assign an attribute value that contains entity references, the user must create an `Attr` node plus any `Text` and `EntityReference` nodes, build the appropriate subtree, and use `setAttributeNode` to assign it as the value of an attribute.

To set an attribute with a qualified name and namespace URI, use the `setAttributeNS` method.

Parameters:

`nameThe` - name of the attribute to create or alter.

`valueValue` - to set in string form.

Throws:

[DOMException](#) - `INVALID_CHARACTER_ERR`: Raised if the specified name contains an illegal character.

`NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is readonly.

removeAttribute

```
public void removeAttribute(java.lang.String name)
    throws DOMException
```

Removes an attribute by name. If the removed attribute is known to have a default value, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

To remove an attribute by local name and namespace URI, use the `removeAttributeNS` method.

Parameters:

`nameThe` - name of the attribute to remove.

Throws:

[DOMException](#) - `NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is readonly.

getAttributeNode

```
public Attr getAttributeNode(java.lang.String name)
```

Retrieves an attribute node by name.

To retrieve an attribute node by qualified name and namespace URI, use the

`getAttributeNodeNS` method.

Parameters:

`nameThe` - name (`nodeName`) of the attribute to retrieve.

Returns:

The `Attr` node with the specified name (`nodeName`) or `null` if there is no such attribute.

setAttributeNode

```
public Attr setAttributeNode(Attr newAttr)
                               throws DOMException
```

Adds a new attribute node. If an attribute with that name (`nodeName`) is already present in the element, it is replaced by the new one.

To add a new attribute node with a qualified name and namespace URI, use the `setAttributeNodeNS` method.

Parameters:

`newAttrThe` - `Attr` node to add to the attribute list.

Returns:

If the `newAttr` attribute replaces an existing attribute, the replaced `Attr` node is returned, otherwise `null` is returned.

Throws:

[DOMException](#) - `WRONG_DOCUMENT_ERR`: Raised if `newAttr` was created from a different document than the one that created the element.

`NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is readonly.

`INUSE_ATTRIBUTE_ERR`: Raised if `newAttr` is already an attribute of another `Element` object. The DOM user must explicitly clone `Attr` nodes to re-use them in other elements.

removeAttributeNode

```
public Attr removeAttributeNode(Attr oldAttr)
                               throws DOMException
```

Removes the specified attribute node. If the removed `Attr` has a default value it is immediately replaced. The replacing attribute has the same namespace URI and local name, as well as the original prefix, when applicable.

Parameters:

`oldAttrThe` - `Attr` node to remove from the attribute list.

Returns:

The `Attr` node that was removed.

Throws:

[DOMException](#) - `NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is readonly.

`NOT_FOUND_ERR`: Raised if `oldAttr` is not an attribute of the element.

getElementsByTagName

```
public NodeList getElementsByTagName(java.lang.String name)
```

Returns a `NodeList` of all descendant `Elements` with a given tag name, in the order in which they are encountered in a preorder traversal of this `Element` tree.

Parameters:

`name` - name of the tag to match on. The special value "*" matches all tags.

Returns:

A list of matching `Element` nodes.

getAttributeNS

```
public java.lang.String getAttributeNS(java.lang.String namespaceURI,  
                                         java.lang.String localName)
```

Retrieves an attribute value by local name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters:

`namespaceURI` - namespace URI of the attribute to retrieve.

`localName` - local name of the attribute to retrieve.

Returns:

The `Attr` value as a string, or the empty string if that attribute does not have a specified or default value.

Since:

DOM Level 2

setAttributeNS

```
public void setAttributeNS(java.lang.String namespaceURI,  
                             java.lang.String qualifiedName,  
                             java.lang.String value)
```

throws [DOMException](#)

Adds a new attribute. If an attribute with the same local name and namespace URI is already present on the element, its prefix is changed to be the prefix part of the `qualifiedName`, and its value is changed to be the `value` parameter. This value is a simple string; it is not parsed as it is being set. So any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately escaped by the implementation when it is written out. In order to assign an attribute value that contains entity references, the user must create an `Attr` node plus any `Text` and `EntityReference` nodes, build the appropriate subtree, and use `setAttributeNodeNS` or `setAttributeNode` to assign it as the value of an attribute. HTML-only DOM implementations do not need to implement this method.

Parameters:

`namespaceURI`The - namespace URI of the attribute to create or alter.

`qualifiedName`The - qualified name of the attribute to create or alter.

`value`The - value to set in string form.

Throws:

[DOMException](#) - `INVALID_CHARACTER_ERR`: Raised if the specified qualified name contains an illegal character.

`NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is readonly.

`NAMESPACE_ERR`: Raised if the `qualifiedName` is malformed, if the `qualifiedName` has a prefix and the `namespaceURI` is null, if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from "http://www.w3.org/XML/1998/namespace", or if the `qualifiedName` is "xmlns" and the `namespaceURI` is different from "http://www.w3.org/2000/xmlns/".

Since:

DOM Level 2

removeAttributeNS

```
public void removeAttributeNS(java.lang.String namespaceURI,
                               java.lang.String localName)
    throws DOMException
```

Removes an attribute by local name and namespace URI. If the removed attribute has a default value it is immediately replaced. The replacing attribute has the same namespace URI and local name, as well as the original prefix.

HTML-only DOM implementations do not need to implement this method.

Parameters:

`namespaceURI`The - namespace URI of the attribute to remove.

`localName`The - local name of the attribute to remove.

Throws:

[DOMException](#) - NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

Since:

DOM Level 2

getAttributeNodeNS

```
public Attr getAttributeNodeNS(java.lang.String namespaceURI,
                                   java.lang.String localName)
```

Retrieves an `Attr` node by local name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters:

`namespaceURI`The - namespace URI of the attribute to retrieve.

`localName`The - local name of the attribute to retrieve.

Returns:

The `Attr` node with the specified attribute local name and namespace URI or `null` if there is no such attribute.

Since:

DOM Level 2

setAttributeNodeNS

```
public Attr setAttributeNodeNS(Attr newAttr)
                                   throws DOMException
```

Adds a new attribute. If an attribute with that local name and that namespace URI is already present in the element, it is replaced by the new one.

HTML-only DOM implementations do not need to implement this method.

Parameters:

`newAttr`The - `Attr` node to add to the attribute list.

Returns:

If the `newAttr` attribute replaces an existing attribute with the same local name and namespace URI, the replaced `Attr` node is returned, otherwise `null` is returned.

Throws:

[DOMException](#) - WRONG_DOCUMENT_ERR: Raised if `newAttr` was created from a different document than the one that created the element.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

INUSE_ATTRIBUTE_ERR: Raised if `newAttr` is already an attribute of another

Element object. The DOM user must explicitly clone `Attr` nodes to re-use them in other elements.

Since:

DOM Level 2

getElementsByTagNameNS

```
public NodeList getElementsByTagNameNS(java.lang.String namespaceURI,  
                                           java.lang.String localName)
```

Returns a `NodeList` of all the descendant `Elements` with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of this `Element` tree. HTML-only DOM implementations do not need to implement this method.

Parameters:

`namespaceURI`The - namespace URI of the elements to match on. The special value "*" matches all namespaces.

`localName`The - local name of the elements to match on. The special value "*" matches all local names.

Returns:

A new `NodeList` object containing all the matched `Elements`.

Since:

DOM Level 2

hasAttribute

```
public boolean hasAttribute(java.lang.String name)
```

Returns `true` when an attribute with a given name is specified on this element or has a default value, `false` otherwise.

Parameters:

`name`The - name of the attribute to look for.

Returns:

`true` if an attribute with the given name is specified on this element or has a default value, `false` otherwise.

Since:

DOM Level 2

hasAttributeNS

```
public boolean hasAttributeNS(java.lang.String namespaceURI,  
                                java.lang.String localName)
```

Returns `true` when an attribute with a given local name and namespace URI is specified on this element or has a default value, `false` otherwise. HTML-only DOM implementations do not need to implement this method.

Parameters:

`namespaceURI`The - namespace URI of the attribute to look for.

`localName`The - local name of the attribute to look for.

Returns:

`true` if an attribute with the given local name and namespace URI is specified or has a default value on this element, `false` otherwise.

Since:

DOM Level 2

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface Entity

public abstract interface **Entity**

extends [Node](#)

This interface represents an entity, either parsed or unparsed, in an XML document. Note that this models the entity itself not the entity declaration. Entity declaration modeling has been left for a later Level of the DOM specification.

The nodeName attribute that is inherited from Node contains the name of the entity.

An XML processor may choose to completely expand entities before the structure model is passed to the DOM; in this case there will be no EntityReference nodes in the document tree.

XML does not mandate that a non-validating XML processor read and process entity declarations made in the external subset or declared in external parameter entities. This means that parsed entities declared in the external subset need not be expanded by some classes of applications, and that the replacement value of the entity may not be available. When the replacement value is available, the corresponding Entity node's child list represents the structure of that replacement text. Otherwise, the child list is empty.

The DOM Level 2 does not support editing Entity nodes; if a user wants to make changes to the contents of an Entity, every related EntityReference node has to be replaced in the structure model by a clone of the Entity's contents, and then the desired changes must be made to each of those clones instead. Entity nodes and all their descendants are readonly.

An Entity node does not have any parent. If the entity contains an unbound namespace prefix, the namespaceURI of the corresponding node in the Entity node subtree is null. The same is true for EntityReference nodes that refer to this entity, when they are created using the createEntityReference method of the Document interface. The DOM Level 2 does not support any mechanism to resolve namespace prefixes.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#), [CDATA_SECTION_NODE](#), [COMMENT_NODE](#),
[DOCUMENT_FRAGMENT_NODE](#), [DOCUMENT_NODE](#), [DOCUMENT_TYPE_NODE](#),
[ELEMENT_NODE](#), [ENTITY_NODE](#), [ENTITY_REFERENCE_NODE](#), [NOTATION_NODE](#),
[PROCESSING_INSTRUCTION_NODE](#), [TEXT_NODE](#)

Method Summary

java.lang.String	getNotationName () For unparsed entities, the name of the notation for the entity.
java.lang.String	getPublicId () The public identifier associated with the entity, if specified.
java.lang.String	getSystemId () The system identifier associated with the entity, if specified.

Methods inherited from interface [org.w3c.dom.Node](#)

[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#),
[getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#),
[getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#),
[getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#),
[hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#),
[replaceChild](#), [setNodeValue](#), [setPrefix](#)

Method Detail

getPublicId

```
public java.lang.String getPublicId( )
```

The public identifier associated with the entity, if specified. If the public identifier was not specified, this is null.

getSystemId

```
public java.lang.String getSystemId( )
```

The system identifier associated with the entity, if specified. If the system identifier was not specified, this is null.

getNotationName

```
public java.lang.String getNotationName( )
```

For unparsed entities, the name of the notation for the entity. For parsed entities, this is null.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface EntityReference

public abstract interface **EntityReference**

extends [Node](#)

EntityReference objects may be inserted into the structure model when an entity reference is in the source document, or when the user wishes to insert an entity reference. Note that character references and references to predefined entities are considered to be expanded by the HTML or XML processor so that characters are represented by their Unicode equivalent rather than by an entity reference. Moreover, the XML processor may completely expand references to entities while building the structure model, instead of providing EntityReference objects. If it does provide such objects, then for a given EntityReference node, it may be that there is no Entity node representing the referenced entity. If such an Entity exists, then the subtree of the EntityReference node is in general a copy of the Entity node subtree. However, this may not be true when an entity contains an unbound namespace prefix. In such a case, because the namespace prefix resolution depends on where the entity reference is, the descendants of the EntityReference node may be bound to different namespace URIs.

As for Entity nodes, EntityReference nodes and all their descendants are readonly.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#), [CDATA_SECTION_NODE](#), [COMMENT_NODE](#),
[DOCUMENT_FRAGMENT_NODE](#), [DOCUMENT_NODE](#), [DOCUMENT_TYPE_NODE](#),
[ELEMENT_NODE](#), [ENTITY_NODE](#), [ENTITY_REFERENCE_NODE](#), [NOTATION_NODE](#),
[PROCESSING_INSTRUCTION_NODE](#), [TEXT_NODE](#)

Methods inherited from interface org.w3c.dom.[Node](#)

[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#),
[getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#),
[getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#),
[getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#),
[hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#),
[replaceChild](#), [setNodeValue](#), [setPrefix](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform

Interface ErrorListener

public abstract interface **ErrorListener**

To provide customized error handling, implement this interface and use the `setErrorListener` method to register an instance of the implementation with the [Transformer](#). The Transformer then reports all errors and warnings through this interface.

If an application does *not* register an `ErrorListener`, errors are reported to `System.err`.

For transformation errors, a Transformer must use this interface instead of throwing an exception: it is up to the application to decide whether to throw an exception for different types of errors and warnings. Note however that the Transformer is not required to continue with the transformation after a call to `fatalError`.

Transformers may use this mechanism to report XML parsing errors as well as transformation errors.

Method Summary

void	error (TransformerException exception) Receive notification of a recoverable error.
void	fatalError (TransformerException exception) Receive notification of a non-recoverable error.
void	warning (TransformerException exception) Receive notification of a warning.

Method Detail

warning

```
public void warning(TransformerException exception)
    throws TransformerException
```

Receive notification of a warning.

[Transformer](#) can use this method to report conditions that are not errors or fatal errors. The default behaviour is to take no action.

After invoking this method, the Transformer must continue with the transformation. It should still be possible for the application to process the document through to the end.

Parameters:

`exception` - The warning information encapsulated in a transformer exception.

Throws:

[TransformerException](#) - if the application chooses to discontinue the transformation.

See Also:

[TransformerException](#)

error

```
public void error(TransformerException exception)
    throws TransformerException
```

Receive notification of a recoverable error.

The transformer must continue to try and provide normal transformation after invoking this method. It should still be possible for the application to process the document through to the end if no other errors are encountered.

Parameters:

`exception` - The error information encapsulated in a transformer exception.

Throws:

[TransformerException](#) - if the application chooses to discontinue the transformation.

See Also:

[TransformerException](#)

fatalError

```
public void fatalError(TransformerException exception)
    throws TransformerException
```

Receive notification of a non-recoverable error.

The transformer must continue to try and provide normal transformation after invoking this method. It should still be possible for the application to process the document through to the end if no other errors are encountered, but there is no guarantee that the output will be useable.

Parameters:

`exception` - The error information encapsulated in a transformer exception.

Throws:

[TransformerException](#) - if the application chooses to discontinue the transformation.

See Also:

[TransformerException](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax.ext

Interface LexicalHandler

All Known Subinterfaces:

[TransformerHandler](#)

public abstract interface **LexicalHandler**

SAX2 extension handler for lexical events.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This is an optional extension handler for SAX2 to provide lexical information about an XML document, such as comments and CDATA section boundaries; XML readers are not required to support this handler, and it is not part of the core SAX2 distribution.

The events in the lexical handler apply to the entire document, not just to the document element, and all lexical handler events must appear between the content handler's startDocument and endDocument events.

To set the LexicalHandler for an XML reader, use the [setProperty](#) method with the propertyId "http://xml.org/sax/properties/lexical-handler". If the reader does not support lexical events, it will throw a [SAXNotRecognizedException](#) or a [SAXNotSupportedException](#) when you attempt to register the handler.

Since:

1.0

Version:

1.0

Author:

David Megginson, sax@megginson.com

See Also:

[XMLReader.setProperty\(java.lang.String, java.lang.Object\)](#),
[SAXNotRecognizedException](#), [SAXNotSupportedException](#)

Method Summary

void	comment (char[] ch, int start, int length) Report an XML comment anywhere in the document.
------	---

void	<code>endCDATA()</code> Report the end of a CDATA section.
void	<code>endDTD()</code> Report the end of DTD declarations.
void	<code>endEntity(java.lang.String name)</code> Report the end of an entity.
void	<code>startCDATA()</code> Report the start of a CDATA section.
void	<code>startDTD(java.lang.String name, java.lang.String publicId, java.lang.String systemId)</code> Report the start of DTD declarations, if any.
void	<code>startEntity(java.lang.String name)</code> Report the beginning of some internal and external XML entities.

Method Detail

startDTD

```
public void startDTD(java.lang.String name,
                     java.lang.String publicId,
                     java.lang.String systemId)
    throws SAXException
```

Report the start of DTD declarations, if any.

This method is intended to report the beginning of the DOCTYPE declaration; if the document has no DOCTYPE declaration, this method will not be invoked.

All declarations reported through [DTDHandler](#) or [DeclHandler](#) events must appear between the `startDTD` and [endDTD](#) events. Declarations are assumed to belong to the internal DTD subset unless they appear between [startEntity](#) and [endEntity](#) events. Comments and processing instructions from the DTD should also be reported between the `startDTD` and `endDTD` events, in their original order of (logical) occurrence; they are not required to appear in their correct locations relative to `DTDHandler` or `DeclHandler` events, however.

Note that the `start/endDTD` events will appear within the `start/endDocument` events from `ContentHandler` and before the first [startElement](#) event.

Parameters:

`name` - The document type name.

`publicId` - The declared public identifier for the external DTD subset, or null if none was

declared.

`systemId` - The declared system identifier for the external DTD subset, or null if none was declared.

Throws:

[SAXException](#) - The application may raise an exception.

See Also:

[endDTD\(\)](#), [startEntity\(java.lang.String\)](#)

endDTD

```
public void endDTD()  
    throws SAXException
```

Report the end of DTD declarations.

This method is intended to report the end of the DOCTYPE declaration; if the document has no DOCTYPE declaration, this method will not be invoked.

Throws:

[SAXException](#) - The application may raise an exception.

See Also:

[startDTD\(java.lang.String, java.lang.String, java.lang.String\)](#)

startEntity

```
public void startEntity(java.lang.String name)  
    throws SAXException
```

Report the beginning of some internal and external XML entities.

The reporting of parameter entities (including the external DTD subset) is optional, and SAX2 drivers that support LexicalHandler may not support it; you can use the <http://xml.org/sax/features/lexical-handler/parameter-entities> feature to query or control the reporting of parameter entities.

General entities are reported with their regular names, parameter entities have '%' prepended to their names, and the external DTD subset has the pseudo-entity name "[dtd]".

When a SAX2 driver is providing these events, all other events must be properly nested within start/end entity events. There is no additional requirement that events from [DeclHandler](#) or [DTDHandler](#) be properly ordered.

Note that skipped entities will be reported through the [skippedEntity](#) event, which is part of the ContentHandler interface.

Because of the streaming event model that SAX uses, some entity boundaries cannot be reported under any circumstances:

- general entities within attribute values
- parameter entities within declarations

These will be silently expanded, with no indication of where the original entity boundaries were.

Note also that the boundaries of character references (which are not really entities anyway) are not reported.

All start/endEntity events must be properly nested.

Parameters:

name - The name of the entity. If it is a parameter entity, the name will begin with '%', and if it is the external DTD subset, it will be "[dtd]".

Throws:

[SAXException](#) - The application may raise an exception.

See Also:

[endEntity\(java.lang.String\)](#),
[DeclHandler.internalEntityDecl\(java.lang.String, java.lang.String\)](#),
[DeclHandler.externalEntityDecl\(java.lang.String, java.lang.String, java.lang.String\)](#)

endEntity

```
public void endEntity(java.lang.String name)
    throws SAXException
```

Report the end of an entity.

Parameters:

name - The name of the entity that is ending.

Throws:

[SAXException](#) - The application may raise an exception.

See Also:

[startEntity\(java.lang.String\)](#)

startCDATA

```
public void startCDATA()  
    throws SAXException
```

Report the start of a CDATA section.

The contents of the CDATA section will be reported through the regular [characters](#) event; this event is intended only to report the boundary.

Throws:

[SAXException](#) - The application may raise an exception.

See Also:

[endCDATA\(\)](#)

endCDATA

```
public void endCDATA()  
    throws SAXException
```

Report the end of a CDATA section.

Throws:

[SAXException](#) - The application may raise an exception.

See Also:

[startCDATA\(\)](#)

comment

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

Report an XML comment anywhere in the document.

This callback will be used for comments inside or outside the document element, including comments in the external DTD subset (if read). Comments in the DTD must be properly nested inside start/endDTD and start/endEntity events (if used).

Parameters:

`ch` - An array holding the characters in the comment.

`start` - The starting position in the array.

`length` - The number of characters to use from the array.

Throws:

[SAXException](#) - The application may raise an exception.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface NamedNodeMap

public abstract interface **NamedNodeMap**

Objects implementing the NamedNodeMap interface are used to represent collections of nodes that can be accessed by name. Note that NamedNodeMap does not inherit from NodeList; NamedNodeMaps are not maintained in any particular order. Objects contained in an object implementing NamedNodeMap may also be accessed by an ordinal index, but this is simply to allow convenient enumeration of the contents of a NamedNodeMap, and does not imply that the DOM specifies an order to these Nodes.

NamedNodeMap objects in the DOM are live.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Method Summary

int	getLength () The number of nodes in this map.
Node	getNamedItem (java.lang.String name) Retrieves a node specified by name.
Node	getNamedItemNS (java.lang.String namespaceURI, java.lang.String localName) Retrieves a node specified by local name and namespace URI.
Node	item (int index) Returns the indexth item in the map.
Node	removeNamedItem (java.lang.String name) Removes a node specified by name.
Node	removeNamedItemNS (java.lang.String namespaceURI, java.lang.String localName) Removes a node specified by local name and namespace URI.
Node	setNamedItem (Node arg) Adds a node using its nodeName attribute.
Node	setNamedItemNS (Node arg) Adds a node using its namespaceURI and localName.

Method Detail

getNamedItem

```
public Node getNamedItem( java.lang.String name )
```

Retrieves a node specified by name.

Parameters:

nameThe - nodeName of a node to retrieve.

Returns:

A Node (of any type) with the specified nodeName, or null if it does not identify any node in this map.

setNamedItem

```
public Node setNamedItem(Node arg)  
    throws DOMException
```

Adds a node using its nodeName attribute. If a node with that name is already present in this map, it is replaced by the new one.

As the nodeName attribute is used to derive the name which the node must be stored under, multiple nodes of certain types (those that have a "special" string value) cannot be stored as the names would clash. This is seen as preferable to allowing nodes to be aliased.

Parameters:

argA - node to store in this map. The node will later be accessible using the value of its nodeName attribute.

Returns:

If the new Node replaces an existing node the replaced Node is returned, otherwise null is returned.

Throws:

[DOMException](#) - WRONG_DOCUMENT_ERR: Raised if arg was created from a different document than the one that created this map.

NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

INUSE_ATTRIBUTE_ERR: Raised if arg is an Attr that is already an attribute of another Element object. The DOM user must explicitly clone Attr nodes to re-use them in other elements.

removeNamedItem

```
public Node removeNamedItem( java.lang.String name)
                               throws DOMException
```

Removes a node specified by name. When this map contains the attributes attached to an element, if the removed attribute is known to have a default value, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

Parameters:

nameThe - nodeName of the node to remove.

Returns:

The node removed from this map if a node with such a name exists.

Throws:

[DOMException](#) - NOT_FOUND_ERR: Raised if there is no node named name in this map.
NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

item

```
public Node item(int index)
```

Returns the indexth item in the map. If index is greater than or equal to the number of nodes in this map, this returns null.

Parameters:

indexIndex - into this map.

Returns:

The node at the indexth position in the map, or null if that is not a valid index.

getLength

```
public int getLength()
```

The number of nodes in this map. The range of valid child node indices is 0 to length-1 inclusive.

getNamedItemNS

```
public Node getNamedItemNS(java.lang.String namespaceURI,
                             java.lang.String localName)
```

Retrieves a node specified by local name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters:

`namespaceURI` - namespace URI of the node to retrieve.

`localName` - local name of the node to retrieve.

Returns:

A [Node](#) (of any type) with the specified local name and namespace URI, or `null` if they do not identify any node in this map.

Since:

DOM Level 2

setNamedItemNS

```
public Node setNamedItemNS(Node arg)
                             throws DOMException
```

Adds a node using its `namespaceURI` and `localName`. If a node with that namespace URI and that local name is already present in this map, it is replaced by the new one. HTML-only DOM implementations do not need to implement this method.

Parameters:

`arg` - node to store in this map. The node will later be accessible using the value of its `namespaceURI` and `localName` attributes.

Returns:

If the new [Node](#) replaces an existing node the replaced [Node](#) is returned, otherwise `null` is returned.

Throws:

[DOMException](#) - `WRONG_DOCUMENT_ERR`: Raised if `arg` was created from a different document than the one that created this map.

`NO_MODIFICATION_ALLOWED_ERR`: Raised if this map is readonly.

`INUSE_ATTRIBUTE_ERR`: Raised if `arg` is an `Attr` that is already an attribute of another `Element` object. The DOM user must explicitly clone `Attr` nodes to re-use them in other elements.

Since:

DOM Level 2

removeNamedItemNS

```
public Node removeNamedItemNS(java.lang.String namespaceURI,
                                   java.lang.String localName)
    throws DOMException
```

Removes a node specified by local name and namespace URI. A removed attribute may be known to have a default value when this map contains the attributes attached to an element, as returned by the `attributes` attribute of the `Node` interface. If so, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

HTML-only DOM implementations do not need to implement this method.

Parameters:

`namespaceURI`The - namespace URI of the node to remove.

`localName`The - local name of the node to remove.

Returns:

The node removed from this map if a node with such a local name and namespace URI exists.

Throws:

[DOMException](#) - NOT_FOUND_ERR: Raised if there is no node with the specified `namespaceURI` and `localName` in this map.

NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

Since:

DOM Level 2

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface NodeList

public abstract interface **NodeList**

The `NodeList` interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented. `NodeList` objects in the DOM are live.

The items in the `NodeList` are accessible via an integral index, starting from 0.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Method Summary

int	getLength () The number of nodes in the list.
Node	item (int index) Returns the <code>index</code> th item in the collection.

Method Detail

item

public [Node](#) **item**(int index)

Returns the `index`th item in the collection. If `index` is greater than or equal to the number of nodes in the list, this returns `null`.

Parameters:

`index` - into the collection.

Returns:

The node at the `index`th position in the `NodeList`, or `null` if that is not a valid index.

getLength

```
public int getLength()
```

The number of nodes in the list. The range of valid child node indices is 0 to `length-1` inclusive.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
SUMMARY: INNER | FIELD | CONSTR | [METHOD](#)DETAIL: FIELD | CONSTR | [METHOD](#)

org.w3c.dom

Interface Notation

public abstract interface **Notation**extends [Node](#)

This interface represents a notation declared in the DTD. A notation either declares, by name, the format of an unparsed entity (see section 4.7 of the XML 1.0 specification), or is used for formal declaration of processing instruction targets (see section 2.6 of the XML 1.0 specification). The nodeName attribute inherited from Node is set to the declared name of the notation.

The DOM Level 1 does not support editing Notation nodes; they are therefore readonly.

A Notation node does not have any parent.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#) , [CDATA_SECTION_NODE](#) , [COMMENT_NODE](#) ,
[DOCUMENT_FRAGMENT_NODE](#) , [DOCUMENT_NODE](#) , [DOCUMENT_TYPE_NODE](#) ,
[ELEMENT_NODE](#) , [ENTITY_NODE](#) , [ENTITY_REFERENCE_NODE](#) , [NOTATION_NODE](#) ,
[PROCESSING_INSTRUCTION_NODE](#) , [TEXT_NODE](#)

Method Summary

java.lang.String	getPublicId () The public identifier of this notation.
java.lang.String	getSystemId () The system identifier of this notation.

Methods inherited from interface org.w3c.dom.[Node](#)

[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#), [getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#), [getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#), [getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#), [hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#), [replaceChild](#), [setNodeValue](#), [setPrefix](#)

Method Detail

getPublicId

```
public java.lang.String getPublicId()
```

The public identifier of this notation. If the public identifier was not specified, this is null.

getSystemId

```
public java.lang.String getSystemId()
```

The system identifier of this notation. If the system identifier was not specified, this is null.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Interface Parser

All Known Implementing Classes:

[XMLReaderAdapter](#)

Deprecated. *This interface has been replaced by the SAX2 [XMLReader](#) interface, which includes Namespace support.*

public abstract interface **Parser**

Basic interface for SAX (Simple API for XML) parsers.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This was the main event supplier interface for SAX1; it has been replaced in SAX2 by [XMLReader](#), which includes Namespace support and sophisticated configurability and extensibility.

All SAX1 parsers must implement this basic interface: it allows applications to register handlers for different types of events and to initiate a parse from a URI, or a character stream.

All SAX1 parsers must also implement a zero-argument constructor (though other constructors are also allowed).

SAX1 parsers are reusable but not re-entrant: the application may reuse a parser object (possibly with a different input source) once the first parse has completed successfully, but it may not invoke the parse() methods recursively within a parse.

Since:

SAX 1.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[EntityResolver](#), [DTDHandler](#), [DocumentHandler](#), [ErrorHandler](#), [HandlerBase](#), [InputSource](#)

Method Summary

void	parse (InputSource source) Deprecated. Parse an XML document.
void	parse (java.lang.String systemId) Deprecated. Parse an XML document from a system identifier (URI).
void	setDocumentHandler (DocumentHandler handler) Deprecated. Allow an application to register a document event handler.
void	setDTDHandler (DTDHandler handler) Deprecated. Allow an application to register a DTD event handler.
void	setEntityResolver (EntityResolver resolver) Deprecated. Allow an application to register a custom entity resolver.
void	setErrorHandler (ErrorHandler handler) Deprecated. Allow an application to register an error event handler.
void	setLocale (java.util.Locale locale) Deprecated. Allow an application to request a locale for errors and warnings.

Method Detail

setLocale

```
public void setLocale(java.util.Locale locale)
    throws SAXException
```

Deprecated.

Allow an application to request a locale for errors and warnings.

SAX parsers are not required to provide localisation for errors and warnings; if they cannot support the requested locale, however, they must throw a SAX exception. Applications may not request a locale change in the middle of a parse.

Parameters:

locale - A Java Locale object.

Throws:

[SAXException](#) - Throws an exception (using the previous or default locale) if the requested locale is not supported.

See Also:

setEntityResolver

```
public void setEntityResolver(EntityResolver resolver)
```

Deprecated.

Allow an application to register a custom entity resolver.

If the application does not register an entity resolver, the SAX parser will resolve system identifiers and open connections to entities itself (this is the default behaviour implemented in `HandlerBase`).

Applications may register a new or different entity resolver in the middle of a parse, and the SAX parser must begin using the new resolver immediately.

Parameters:

`resolver` - The object for resolving entities.

See Also:

[EntityResolver](#), [HandlerBase](#)

setDTDHandler

```
public void setDTDHandler(DTDHandler handler)
```

Deprecated.

Allow an application to register a DTD event handler.

If the application does not register a DTD handler, all DTD events reported by the SAX parser will be silently ignored (this is the default behaviour implemented by `HandlerBase`).

Applications may register a new or different handler in the middle of a parse, and the SAX parser must begin using the new handler immediately.

Parameters:

`handler` - The DTD handler.

See Also:

[DTDHandler](#), [HandlerBase](#)

setDocumentHandler

```
public void setDocumentHandler(DocumentHandler handler)
```

Deprecated.

Allow an application to register a document event handler.

If the application does not register a document handler, all document events reported by the SAX parser will be silently ignored (this is the default behaviour implemented by `HandlerBase`).

Applications may register a new or different handler in the middle of a parse, and the SAX parser must begin using the new handler immediately.

Parameters:

handler - The document handler.

See Also:

[DocumentHandler](#), [HandlerBase](#)

setErrorHandler

```
public void setErrorHandler(ErrorHandler handler)
```

Deprecated.

Allow an application to register an error event handler.

If the application does not register an error event handler, all error events reported by the SAX parser will be silently ignored, except for `fatalError`, which will throw a `SAXException` (this is the default behaviour implemented by `HandlerBase`).

Applications may register a new or different handler in the middle of a parse, and the SAX parser must begin using the new handler immediately.

Parameters:

handler - The error handler.

See Also:

[ErrorHandler](#), [SAXException](#), [HandlerBase](#)

parse

```
public void parse(InputSource source)  
    throws SAXException,  
           java.io.IOException
```

Deprecated.

Parse an XML document.

The application can use this method to instruct the SAX parser to begin parsing an XML document from any valid input source (a character stream, a byte stream, or a URI).

Applications may not invoke this method while a parse is in progress (they should create a new Parser instead for each additional XML document). Once a parse is complete, an application may reuse the same Parser object, possibly with a different input source.

Parameters:

`source` - The input source for the top-level of the XML document.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

`java.io.IOException` - An IO exception from the parser, possibly from a byte stream or character stream supplied by the application.

See Also:

[InputSource](#), [parse\(java.lang.String\)](#),
[setEntityResolver\(org.xml.sax.EntityResolver\)](#),
[setDTDHandler\(org.xml.sax.DTDHandler\)](#),
[setDocumentHandler\(org.xml.sax.DocumentHandler\)](#),
[setErrorHandler\(org.xml.sax.ErrorHandler\)](#)

parse

```
public void parse(java.lang.String systemId)
    throws SAXException,
           java.io.IOException
```

Deprecated.

Parse an XML document from a system identifier (URI).

This method is a shortcut for the common case of reading a document from a system identifier. It is the exact equivalent of the following:

```
parse(new InputSource(systemId));
```

If the system identifier is a URL, it must be fully resolved by the application before it is passed to the parser.

Parameters:

`systemId` - The system identifier (URI).

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

[java.io.IOException](#) - An IO exception from the parser, possibly from a byte stream or character stream supplied by the application.

See Also:

[parse\(org.xml.sax.InputSource\)](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface ProcessingInstruction

public abstract interface **ProcessingInstruction**

extends [Node](#)

The ProcessingInstruction interface represents a "processing instruction", used in XML as a way to keep processor-specific information in the text of the document.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#), [CDATA_SECTION_NODE](#), [COMMENT_NODE](#),
[DOCUMENT_FRAGMENT_NODE](#), [DOCUMENT_NODE](#), [DOCUMENT_TYPE_NODE](#),
[ELEMENT_NODE](#), [ENTITY_NODE](#), [ENTITY_REFERENCE_NODE](#), [NOTATION_NODE](#),
[PROCESSING_INSTRUCTION_NODE](#), [TEXT_NODE](#)

Method Summary

java.lang.String	getData () The content of this processing instruction.
java.lang.String	getTarget () The target of this processing instruction.
void	setData (java.lang.String data)

Methods inherited from interface org.w3c.dom.[Node](#)

[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#),
[getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#),
[getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#),
[getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#),
[hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#),
[replaceChild](#), [setNodeValue](#), [setPrefix](#)

Method Detail

getTarget

```
public java.lang.String getTarget()
```

The target of this processing instruction. XML defines this as being the first token following the markup that begins the processing instruction.

getData

```
public java.lang.String getData()
```

The content of this processing instruction. This is from the first non white space character after the target to the character immediately preceding the ?>.

Throws:

[DOMException](#) - NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

setData

```
public void setData(java.lang.String data)  
    throws DOMException
```

[Overview](#) [Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Class SAXException

```

java.lang.Object
|
+-- java.lang.Throwable
    |
    +-- java.lang.Exception
        |
        +-- org.xml.sax.SAXException
  
```

Direct Known Subclasses:

[SAXNotRecognizedException](#), [SAXNotSupportedException](#), [SAXParseException](#)

```
public class SAXException
```

```
extends java.lang.Exception
```

Encapsulate a general SAX error or warning.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This class can contain basic error or warning information from either the XML parser or the application: a parser writer or application writer can subclass it to provide additional functionality. SAX handlers may throw this exception or any exception subclassed from it.

If the application needs to pass through other types of exceptions, it must wrap those exceptions in a SAXException or an exception derived from a SAXException.

If the parser or application needs to include information about a specific location in an XML document, it should use the [SAXParseException](#) subclass.

Since:

SAX 1.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[SAXParseException](#), [Serialized Form](#)

Constructor Summary

[**SAXException**](#)(java.lang.Exception e)

Create a new SAXException wrapping an existing exception.

[**SAXException**](#)(java.lang.String message)

Create a new SAXException.

[**SAXException**](#)(java.lang.String message, java.lang.Exception e)

Create a new SAXException from an existing exception.

Method Summary

java.lang.Exception	getException ()
java.lang.String	getMessage ()
java.lang.String	toString ()

Return the embedded exception, if any.

Return a detail message for this exception.

Override toString to pick up any embedded exception.

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, printStackTrace, printStackTrace, printStackTrace

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

SAXException

public **SAXException**(java.lang.String message)

Create a new SAXException.

Parameters:

message - The error or warning message.

See Also:

[Parser.setLocale\(java.util.Locale\)](#)

SAXException

```
public SAXException(java.lang.Exception e)
```

Create a new SAXException wrapping an existing exception.

The existing exception will be embedded in the new one, and its message will become the default message for the SAXException.

Parameters:

e - The exception to be wrapped in a SAXException.

SAXException

```
public SAXException(java.lang.String message,  
                    java.lang.Exception e)
```

Create a new SAXException from an existing exception.

The existing exception will be embedded in the new one, but the new exception will have its own message.

Parameters:

message - The detail message.

e - The exception to be wrapped in a SAXException.

See Also:

[Parser.setLocale\(java.util.Locale\)](#)

Method Detail

getMessage

```
public java.lang.String getMessage()
```

Return a detail message for this exception.

If there is an embedded exception, and if the SAXException has no detail message of its own, this method will return the detail message from the embedded exception.

Returns:

The error or warning message.

Overrides:

getMessage in class java.lang.Throwable

See Also:

[Parser.setLocale\(java.util.Locale\)](#)

getException

```
public java.lang.Exception getException()
```

Return the embedded exception, if any.

Returns:

The embedded exception, or null if there is none.

toString

```
public java.lang.String toString()
```

Override toString to pick up any embedded exception.

Returns:

A string representation of this exception.

Overrides:

toString in class java.lang.Throwable

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Class SAXNotRecognizedException

```

java.lang.Object
|
+-- java.lang.Throwable
    |
    +-- java.lang.Exception
        |
        +-- org.xml.sax.SAXException
            |
            +-- org.xml.sax.SAXNotRecognizedException
  
```

public class **SAXNotRecognizedException**

extends [SAXException](#)

Exception class for an unrecognized identifier.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

An XMLReader will throw this exception when it finds an unrecognized feature or property identifier; SAX applications and extensions may use this class for other, similar purposes.

Since:

SAX 2.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[SAXNotSupportedException](#), [Serialized Form](#)

Constructor Summary

[SAXNotRecognizedException](#)(java.lang.String message)

Construct a new exception with the given message.

Methods inherited from class org.xml.sax.[SAXException](#)[getException](#), [getMessage](#), [toString](#)**Methods inherited from class java.lang.Throwable**[fillInStackTrace](#), [getLocalizedMessage](#), [printStackTrace](#),
[printStackTrace](#), [printStackTrace](#)**Methods inherited from class java.lang.Object**[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [wait](#),
[wait](#), [wait](#)

Constructor Detail

SAXNotRecognizedException

```
public SAXNotRecognizedException(java.lang.String message)
```

Construct a new exception with the given message.

Parameters:

message - The text message of the exception.

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Class SAXNotSupportedException

```

java.lang.Object
|
+-- java.lang.Throwable
    |
    +-- java.lang.Exception
        |
        +-- org.xml.sax.SAXException
            |
            +-- org.xml.sax.SAXNotSupportedException
  
```

public class **SAXNotSupportedException**

extends [SAXException](#)

Exception class for an unsupported operation.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

An XMLReader will throw this exception when it recognizes a feature or property identifier, but cannot perform the requested operation (setting a state or value). Other SAX2 applications and extensions may use this class for similar purposes.

Since:

SAX 2.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[SAXNotRecognizedException](#), [Serialized Form](#)

Constructor Summary

[SAXNotSupportedException](#)(java.lang.String message)

Construct a new exception with the given message.

Methods inherited from class org.xml.sax.[SAXException](#)[getException](#), [getMessage](#), [toString](#)**Methods inherited from class java.lang.Throwable**[fillInStackTrace](#), [getLocalizedMessage](#), [printStackTrace](#),
[printStackTrace](#), [printStackTrace](#)**Methods inherited from class java.lang.Object**[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [wait](#),
[wait](#), [wait](#)

Constructor Detail

SAXNotSupportedException

```
public SAXNotSupportedException(java.lang.String message)
```

Construct a new exception with the given message.

Parameters:

message - The text message of the exception.

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)

 SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

 DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Class SAXParseException

```

java.lang.Object
|
+-- java.lang.Throwable
    |
    +-- java.lang.Exception
        |
        +-- org.xml.sax.SAXException
            |
            +-- org.xml.sax.SAXParseException
  
```

public class **SAXParseException**

extends [SAXException](#)

Encapsulate an XML parse error or warning.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This exception will include information for locating the error in the original XML document. Note that although the application will receive a SAXParseException as the argument to the handlers in the [ErrorHandler](#) interface, the application is not actually required to throw the exception; instead, it can simply read the information in it and take a different action.

Since this exception is a subclass of [SAXException](#), it inherits the ability to wrap another exception.

Since:

SAX 1.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[SAXException](#), [Locator](#), [ErrorHandler](#), [Serialized Form](#)

Constructor Summary

[**SAXParseException**](#)(java.lang.String message, [Locator](#) locator)

Create a new SAXParseException from a message and a Locator.

[**SAXParseException**](#)(java.lang.String message, [Locator](#) locator, java.lang.Exception e)

Wrap an existing exception in a SAXParseException.

[**SAXParseException**](#)(java.lang.String message, java.lang.String publicId, java.lang.String systemId, int lineNumber, int columnNumber)

Create a new SAXParseException.

[**SAXParseException**](#)(java.lang.String message, java.lang.String publicId, java.lang.String systemId, int lineNumber, int columnNumber, java.lang.Exception e)

Create a new SAXParseException with an embedded exception.

Method Summary

int	getColumnNumber ()	The column number of the end of the text where the exception occurred.
int	getLineNumber ()	The line number of the end of the text where the exception occurred.
java.lang.String	getPublicId ()	Get the public identifier of the entity where the exception occurred.
java.lang.String	getSystemId ()	Get the system identifier of the entity where the exception occurred.

Methods inherited from class org.xml.sax.[SAXException](#)

[getException](#), [getMessage](#), [toString](#)

Methods inherited from class java.lang.Throwable

[fillInStackTrace](#), [getLocalizedMessage](#), [printStackTrace](#), [printStackTrace](#), [printStackTrace](#)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait
--

Constructor Detail

SAXParseException

```
public SAXParseException(java.lang.String message,
                          Locator locator)
```

Create a new SAXParseException from a message and a Locator.

This constructor is especially useful when an application is creating its own exception from within a [ContentHandler](#) callback.

Parameters:

message - The error or warning message.

locator - The locator object for the error or warning (may be null).

See Also:

[Locator](#), [Parser.setLocale\(java.util.Locale\)](#)

SAXParseException

```
public SAXParseException(java.lang.String message,
                          Locator locator,
                          java.lang.Exception e)
```

Wrap an existing exception in a SAXParseException.

This constructor is especially useful when an application is creating its own exception from within a [ContentHandler](#) callback, and needs to wrap an existing exception that is not a subclass of [SAXException](#).

Parameters:

message - The error or warning message, or null to use the message from the embedded exception.

locator - The locator object for the error or warning (may be null).

e - Any exception.

See Also:

[Locator](#), [Parser.setLocale\(java.util.Locale\)](#)

SAXParseException

```
public SAXParseException( java.lang.String message,
                          java.lang.String publicId,
                          java.lang.String systemId,
                          int lineNumber,
                          int columnNumber )
```

Create a new SAXParseException.

This constructor is most useful for parser writers.

If the system identifier is a URL, the parser must resolve it fully before creating the exception.

Parameters:

message - The error or warning message.

publicId - The public identifier of the entity that generated the error or warning.

systemId - The system identifier of the entity that generated the error or warning.

lineNumber - The line number of the end of the text that caused the error or warning.

columnNumber - The column number of the end of the text that cause the error or warning.

See Also:

[Parser.setLocale\(java.util.Locale\)](#)

SAXParseException

```
public SAXParseException( java.lang.String message,
                          java.lang.String publicId,
                          java.lang.String systemId,
                          int lineNumber,
                          int columnNumber,
                          java.lang.Exception e )
```

Create a new SAXParseException with an embedded exception.

This constructor is most useful for parser writers who need to wrap an exception that is not a subclass of [SAXException](#).

If the system identifier is a URL, the parser must resolve it fully before creating the exception.

Parameters:

message - The error or warning message, or null to use the message from the embedded exception.

publicId - The public identifier of the entity that generated the error or warning.

`systemId` - The system identifier of the entity that generated the error or warning.

`lineNumber` - The line number of the end of the text that caused the error or warning.

`columnNumber` - The column number of the end of the text that caused the error or warning.

`e` - Another exception to embed in this one.

See Also:

[Parser.setLocale\(java.util.Locale\)](#)

Method Detail

getPublicId

```
public java.lang.String getPublicId()
```

Get the public identifier of the entity where the exception occurred.

Returns:

A string containing the public identifier, or null if none is available.

See Also:

[Locator.getPublicId\(\)](#)

getSystemId

```
public java.lang.String getSystemId()
```

Get the system identifier of the entity where the exception occurred.

If the system identifier is a URL, it will be resolved fully.

Returns:

A string containing the system identifier, or null if none is available.

See Also:

[Locator.getSystemId\(\)](#)

getLineNumber

```
public int getLineNumber()
```

The line number of the end of the text where the exception occurred.

Returns:

An integer representing the line number, or -1 if none is available.

See Also:[Locator.getLineNumber\(\)](#)

getColumnNumber

```
public int getColumnNumber()
```

The column number of the end of the text where the exception occurred.

The first column in a line is position 1.

Returns:

An integer representing the column number, or -1 if none is available.

See Also:[Locator.getColumnNumber\(\)](#)

Overview	Package	Class	Use	Tree	Deprecated	Index	Help
--------------------------	-------------------------	--------------	---------------------	----------------------	----------------------------	-----------------------	----------------------

[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform.sax

Class SAXTransformerFactory

java.lang.Object

```

|
+-- javax.xml.transform.TransformerFactory

```

```

|
+-- javax.xml.transform.sax.SAXTransformerFactory

```

public abstract class **SAXTransformerFactory**extends [TransformerFactory](#)

This class extends TransformerFactory to provide SAX-specific factory methods. It provides two types of ContentHandlers, one for creating Transformers, the other for creating Templates objects.

If an application wants to set the ErrorHandler or EntityResolver for an XMLReader used during a transformation, it should use a URIResolver to return the SAXSource which provides (with getXMLReader) a reference to the XMLReader.

Field Summary

static java.lang.String	FEATURE If TransformerFactory.getFeature(java.lang.String) returns true when passed this value as an argument, the TransformerFactory returned from TransformerFactory.newInstance() may be safely cast to a SAXTransformerFactory.
static java.lang.String	FEATURE_XMLFILTER If TransformerFactory.getFeature(java.lang.String) returns true when passed this value as an argument, the newXMLFilter(Source src) and newXMLFilter(Templates templates) methods are supported.

Constructor Summary

protected	SAXTransformerFactory() The default constructor is protected on purpose.
-----------	---

Method Summary

abstract TemplatesHandler	newTemplatesHandler () Get a TemplatesHandler object that can process SAX ContentHandler events into a Templates object.
abstract TransformerHandler	newTransformerHandler () Get a TransformerHandler object that can process SAX ContentHandler events into a Result.
abstract TransformerHandler	newTransformerHandler (Source src) Get a TransformerHandler object that can process SAX ContentHandler events into a Result, based on the transformation instructions specified by the argument.
abstract TransformerHandler	newTransformerHandler (Templates templates) Get a TransformerHandler object that can process SAX ContentHandler events into a Result, based on the Templates argument.
abstract XMLFilter	newXMLFilter (Source src) Create an XMLFilter that uses the given Source as the transformation instructions.
abstract XMLFilter	newXMLFilter (Templates templates) Create an XMLFilter, based on the Templates argument..

Methods inherited from class [javax.xml.transform.TransformerFactory](#)

[getAssociatedStylesheet](#), [getAttribute](#), [getErrorListener](#), [getFeature](#), [getURIResolver](#), [newInstance](#), [newTemplates](#), [newTransformer](#), [newTransformer](#), [setAttribute](#), [setErrorListener](#), [setURIResolver](#)

Methods inherited from class [java.lang.Object](#)

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#), [wait](#)

Field Detail

FEATURE

```
public static final java.lang.String FEATURE
```

If [TransformerFactory.getFeature\(java.lang.String\)](#) returns true when passed this value as an argument, the TransformerFactory returned from [TransformerFactory.newInstance\(\)](#) may be safely cast to a SAXTransformerFactory.

FEATURE_XMLFILTER

public static final java.lang.String **FEATURE_XMLFILTER**

If [TransformerFactory.getFeature\(java.lang.String\)](#) returns true when passed this value as an argument, the [newXMLFilter\(Source src\)](#) and [newXMLFilter\(Templates templates\)](#) methods are supported.

Constructor Detail

SAXTransformerFactory

protected **SAXTransformerFactory**()

The default constructor is protected on purpose.

Method Detail

newTransformerHandler

public abstract [TransformerHandler](#) **newTransformerHandler**([Source](#) src)
throws

[TransformerConfigurationException](#)

Get a TransformerHandler object that can process SAX ContentHandler events into a Result, based on the transformation instructions specified by the argument.

Parameters:

src - The Source of the transformation instructions.

Returns:

TransformerHandler ready to transform SAX events.

Throws:

[TransformerConfigurationException](#) - If for some reason the TransformerHandler can not be created.

newTransformerHandler

public abstract [TransformerHandler](#) **newTransformerHandler**([Templates](#) templates)
throws

[TransformerConfigurationException](#)

Get a TransformerHandler object that can process SAX ContentHandler events into a Result, based on the Templates argument.

Parameters:

templates - The compiled transformation instructions.

Returns:

TransformerHandler ready to transform SAX events.

Throws:

[TransformerConfigurationException](#) - If for some reason the TransformerHandler can not be created.

newTransformerHandler

```
public abstract TransformerHandler newTransformerHandler()
                                     throws
```

[TransformerConfigurationException](#)

Get a TransformerHandler object that can process SAX ContentHandler events into a Result. The transformation is defined as an identity (or copy) transformation, for example to copy a series of SAX parse events into a DOM tree.

Returns:

A non-null reference to a TransformerHandler, that may be used as a ContentHandler for SAX parse events.

Throws:

[TransformerConfigurationException](#) - If for some reason the TransformerHandler cannot be created.

newTemplatesHandler

```
public abstract TemplatesHandler newTemplatesHandler()
                                     throws
```

[TransformerConfigurationException](#)

Get a TemplatesHandler object that can process SAX ContentHandler events into a Templates object.

Returns:

A non-null reference to a TransformerHandler, that may be used as a ContentHandler for SAX parse events.

Throws:

[TransformerConfigurationException](#) - If for some reason the TemplatesHandler cannot be created.

newXMLFilter

```
public abstract XMLFilter newXMLFilter(Source src)
                                     throws TransformerConfigurationException
```

Create an XMLFilter that uses the given Source as the transformation instructions.

Parameters:

src - The Source of the transformation instructions.

Returns:

An XMLFilter object, or null if this feature is not supported.

Throws:

[TransformerConfigurationException](#) - If for some reason the TemplatesHandler cannot be created.

newXMLFilter

```
public abstract XMLFilter newXMLFilter(Templates templates)  
                                   throws TransformerConfigurationException
```

Create an XMLFilter, based on the Templates argument..

Parameters:

templates - The compiled transformation instructions.

Returns:

An XMLFilter object, or null if this feature is not supported.

Throws:

[TransformerConfigurationException](#) - If for some reason the TemplatesHandler cannot be created.

[Overview](#) [Package](#) **[Class](#)** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform

Interface SourceLocator

All Known Subinterfaces:

[DOMLocator](#)

public abstract interface **SourceLocator**

This interface is primarily for the purposes of reporting where an error occurred in the XML source or transformation instructions.

Method Summary

int	getColumnNumber () Return the character position where the current document event ends.
int	getLineNumber () Return the line number where the current document event ends.
java.lang.String	getPublicId () Return the public identifier for the current document event.
java.lang.String	getSystemId () Return the system identifier for the current document event.

Method Detail

getPublicId

```
public java.lang.String getPublicId( )
```

Return the public identifier for the current document event.

The return value is the public identifier of the document entity or of the external parsed entity in which the markup that triggered the event appears.

Returns:

A string containing the public identifier, or null if none is available.

See Also:

[getSystemId\(\)](#)

getSystemId

```
public java.lang.String getSystemId()
```

Return the system identifier for the current document event.

The return value is the system identifier of the document entity or of the external parsed entity in which the markup that triggered the event appears.

If the system identifier is a URL, the parser must resolve it fully before passing it to the application.

Returns:

A string containing the system identifier, or null if none is available.

See Also:

[getPublicId\(\)](#)

getLineNumber

```
public int getLineNumber()
```

Return the line number where the current document event ends.

Warning: The return value from the method is intended only as an approximation for the sake of error reporting; it is not intended to provide sufficient information to edit the character content of the original XML document.

The return value is an approximation of the line number in the document entity or external parsed entity where the markup that triggered the event appears.

Returns:

The line number, or -1 if none is available.

See Also:

[getColumnNumber\(\)](#)

getColumnNumber

```
public int getColumnNumber()
```

Return the character position where the current document event ends.

Warning: The return value from the method is intended only as an approximation for the sake of error reporting; it is not intended to provide sufficient information to edit the character content of the original XML document.

The return value is an approximation of the column number in the document entity or external parsed entity where the markup that triggered the event appears.

Returns:

The column number, or -1 if none is available.

See Also:

[getLineNumber\(\)](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform.stream

Class StreamSource

java.lang.Object

|

+-- **javax.xml.transform.stream.StreamSource**public class **StreamSource**

extends java.lang.Object

implements [Source](#)

Acts as an holder for a transformation Source in the form of a stream of XML markup.

Field Summary

static java.lang.String	FEATURE
	If TransformerFactory.getFeature(java.lang.String) returns true when passed this value as an argument, the Transformer supports Source input of this type.

Constructor Summary

[StreamSource](#)()

Zero-argument default constructor.

[StreamSource](#)(java.io.File f)

Construct a StreamSource from a File.

[StreamSource](#)(java.io.InputStream inputStream)

Construct a StreamSource from a byte stream.

[StreamSource](#)(java.io.InputStream inputStream,
java.lang.String systemId)

Construct a StreamSource from a byte stream.

[StreamSource](#)(java.io.Reader reader)

Construct a StreamSource from a character reader.

[StreamSource](#)(java.io.Reader reader, java.lang.String systemId)

Construct a StreamSource from a character reader.

[StreamSource](#)(java.lang.String systemId)

Construct a StreamSource from a URL.

Method Summary

java.io.InputStream	<u>getInputStream</u> () Get the byte stream that was set with setByteStream.
java.lang.String	<u>getPublicId</u> () Get the public identifier that was set with setPublicId.
java.io.Reader	<u>getReader</u> () Get the character stream that was set with setReader.
java.lang.String	<u>getSystemId</u> () Get the system identifier that was set with setSystemId.
void	<u>setInputStream</u> (java.io.InputStream inputStream) Set the byte stream to be used as input.
void	<u>setPublicId</u> (java.lang.String publicId) Set the public identifier for this Source.
void	<u>setReader</u> (java.io.Reader reader) Set the input to be a character reader.
void	<u>setSystemId</u> (java.io.File f) Set the system ID from a File reference.
void	<u>setSystemId</u> (java.lang.String systemId) Set the system identifier for this Source.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

FEATURE

```
public static final java.lang.String FEATURE
```

If [TransformerFactory.getFeature\(java.lang.String\)](#) returns true when passed this value as an argument, the Transformer supports Source input of this type.

Constructor Detail

StreamSource

```
public StreamSource()
```

Zero-argument default constructor. If this constructor is used, and no other method is called, the transformer will assume an empty input tree, with a default root node.

StreamSource

```
public StreamSource(java.io.InputStream inputStream)
```

Construct a StreamSource from a byte stream. Normally, a stream should be used rather than a reader, so the XML parser can resolve character encoding specified by the XML declaration.

If this constructor is used to process a stylesheet, normally `setSystemId` should also be called, so that relative URI references can be resolved.

Parameters:

`inputStream` - A valid `InputStream` reference to an XML stream.

StreamSource

```
public StreamSource(java.io.InputStream inputStream,  
                    java.lang.String systemId)
```

Construct a StreamSource from a byte stream. Normally, a stream should be used rather than a reader, so that the XML parser can resolve character encoding specified by the XML declaration.

This constructor allows the `systemID` to be set in addition to the input stream, which allows relative URIs to be processed.

Parameters:

`inputStream` - A valid `InputStream` reference to an XML stream.

`systemId` - Must be a `String` that conforms to the URI syntax.

StreamSource

```
public StreamSource(java.io.Reader reader)
```

Construct a StreamSource from a character reader. Normally, a stream should be used rather than a reader, so that the XML parser can resolve character encoding specified by the XML declaration. However, in many cases the encoding of the input stream is already resolved, as in the case of reading XML from a StringReader.

Parameters:

reader - A valid Reader reference to an XML character stream.

StreamSource

```
public StreamSource(java.io.Reader reader,  
                    java.lang.String systemId)
```

Construct a StreamSource from a character reader. Normally, a stream should be used rather than a reader, so that the XML parser may resolve character encoding specified by the XML declaration. However, in many cases the encoding of the input stream is already resolved, as in the case of reading XML from a StringReader.

Parameters:

reader - A valid Reader reference to an XML character stream.

systemId - Must be a String that conforms to the URI syntax.

StreamSource

```
public StreamSource(java.lang.String systemId)
```

Construct a StreamSource from a URL.

Parameters:

systemId - Must be a String that conforms to the URI syntax.

StreamSource

```
public StreamSource(java.io.File f)
```

Construct a StreamSource from a File.

Parameters:

f - Must a non-null File reference.

Method Detail

setInputStream

```
public void setInputStream(java.io.InputStream inputStream)
```

Set the byte stream to be used as input. Normally, a stream should be used rather than a reader, so that the XML parser can resolve character encoding specified by the XML declaration.

If this Source object is used to process a stylesheet, normally `setSystemId` should also be called, so that relative URL references can be resolved.

Parameters:

`inputStream` - A valid `InputStream` reference to an XML stream.

getInputStream

```
public java.io.InputStream getInputStream()
```

Get the byte stream that was set with `setByteStream`.

Returns:

The byte stream that was set with `setByteStream`, or null if `setByteStream` or the `ByteStream` constructor was not called.

setReader

```
public void setReader(java.io.Reader reader)
```

Set the input to be a character reader. Normally, a stream should be used rather than a reader, so that the XML parser can resolve character encoding specified by the XML declaration. However, in many cases the encoding of the input stream is already resolved, as in the case of reading XML from a `StringReader`.

Parameters:

`reader` - A valid `Reader` reference to an XML `CharacterStream`.

getReader

```
public java.io.Reader getReader()
```

Get the character stream that was set with `setReader`.

Returns:

The character stream that was set with `setReader`, or null if `setReader` or the `Reader` constructor was not called.

setPublicId

```
public void setPublicId(java.lang.String publicId)
```

Set the public identifier for this Source.

The public identifier is always optional: if the application writer includes one, it will be provided as part of the location information.

Parameters:

`publicId` - The public identifier as a string.

getPublicId

```
public java.lang.String getPublicId()
```

Get the public identifier that was set with `setPublicId`.

Returns:

The public identifier that was set with `setPublicId`, or null if `setPublicId` was not called.

setSystemId

```
public void setSystemId(java.lang.String systemId)
```

Set the system identifier for this Source.

The system identifier is optional if there is a byte stream or a character stream, but it is still useful to provide one, since the application can use it to resolve relative URIs and can include it in error messages and warnings (the parser will attempt to open a connection to the URI only if there is no byte stream or character stream specified).

Specified by:

[setSystemId](#) in interface [Source](#)

Parameters:

`systemId` - The system identifier as a URL string.

getSystemId

```
public java.lang.String getSystemId()
```

Get the system identifier that was set with `setSystemId`.

Specified by:

[getSystemId](#) in interface [Source](#)

Returns:

The system identifier that was set with `setSystemId`, or null if `setSystemId` was not called.

setSystemId

```
public void setSystemId(java.io.File f)
```

Set the system ID from a File reference.

Parameters:

`f` - Must a non-null File reference.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform

Interface Templates

public abstract interface **Templates**

An object that implements this interface is the runtime representation of processed transformation instructions.

Templates must be threadsafe for a given instance over multiple threads running concurrently, and may be used multiple times in a given session.

Method Summary

java.util.Properties	getOutputProperties () Get the static properties for xsl:output.
Transformer	newTransformer () Create a new transformation context for this Templates object.

Method Detail

newTransformer

```
public Transformer newTransformer( )
    throws TransformerConfigurationException
```

Create a new transformation context for this Templates object.

Returns:

A valid non-null instance of a Transformer.

Throws:

[TransformerConfigurationException](#) - if a Transformer can not be created.

getOutputProperties

```
public java.util.Properties getOutputProperties()
```

Get the static properties for `xsl:output`. The object returned will be a clone of the internal values. Accordingly, it can be mutated without mutating the Templates object, and then handed in to [Transformer.setOutputProperties\(java.util.Properties\)](#).

The properties returned should contain properties set by the stylesheet, and these properties are "defaulted" by default properties specified by [section 16 of the XSL Transformations \(XSLT\) W3C Recommendation](#). The properties that were specifically set by the stylesheet should be in the base Properties list, while the XSLT default properties that were not specifically set should be in the "default" Properties list. Thus, `getOutputProperties().getProperty(String key)` will obtain any property in that was set by the stylesheet, *or* the default properties, while `getOutputProperties().get(String key)` will only retrieve properties that were explicitly set in the stylesheet.

For XSLT, [Attribute Value Templates](#) attribute values will be returned unexpanded (since there is no context at this point). The namespace prefixes inside Attribute Value Templates will be unexpanded, so that they remain valid XPath values. (For XSLT 1.0, this is not a problem since Attribute Value Templates are not allowed for `xsl:output` attributes. However, they will be allowed in versions after 1.1.)

Returns:

A Properties object, never null.

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform.sax

Interface TemplatesHandler

public abstract interface **TemplatesHandler**extends [ContentHandler](#)

A SAX ContentHandler that may be used to process SAX parse events (parsing transformation instructions) into a Templates object.

Note that TemplatesHandler does not need to implement LexicalHandler.

Method Summary

java.lang.String	getSystemId () Get the base ID (URI or system ID) from where relative URLs will be resolved.
Templates	getTemplates () When a TemplatesHandler object is used as a ContentHandler for the parsing of transformation instructions, it creates a Templates object, which the caller can get once the SAX events have been completed.
void	setSystemId (java.lang.String systemID) Set the base ID (URI or system ID) for the Templates object created by this builder.

Methods inherited from interface org.xml.sax.[ContentHandler](#)

[characters](#), [endDocument](#), [endElement](#), [endPrefixMapping](#),
[ignorableWhitespace](#), [processingInstruction](#), [setDocumentLocator](#),
[skippedEntity](#), [startDocument](#), [startElement](#), [startPrefixMapping](#)

Method Detail

getTemplates

```
public Templates getTemplates( )
```

When a TemplatesHandler object is used as a ContentHandler for the parsing of transformation instructions, it creates a Templates object, which the caller can get once the SAX events have been completed.

Returns:

The Templates object that was created during the SAX event process, or null if no Templates object has been created.

setSystemId

```
public void setSystemId(java.lang.String systemID)
```

Set the base ID (URI or system ID) for the Templates object created by this builder. This must be set in order to resolve relative URIs in the stylesheet. This must be called before the startDocument event.

Parameters:

baseID - Base URI for this stylesheet.

getSystemId

```
public java.lang.String getSystemId( )
```

Get the base ID (URI or system ID) from where relative URLs will be resolved.

Returns:

The systemID that was set with [setSystemId\(java.lang.String \)](#).

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.w3c.dom

Interface Text

All Known Subinterfaces:

[CDATASection](#)
public abstract interface **Text**extends [CharacterData](#)

The `Text` interface inherits from `CharacterData` and represents the textual content (termed character data in XML) of an `Element` or `Attr`. If there is no markup inside an element's content, the text is contained in a single object implementing the `Text` interface that is the only child of the element. If there is markup, it is parsed into the information items (elements, comments, etc.) and `Text` nodes that form the list of children of the element.

When a document is first made available via the DOM, there is only one `Text` node for each block of text. Users may create adjacent `Text` nodes that represent the contents of a given element without any intervening markup, but should be aware that there is no way to represent the separations between these nodes in XML or HTML, so they will not (in general) persist between DOM editing sessions. The `normalize()` method on `Node` merges any such adjacent `Text` objects into a single node for each block of text.

See also the [Document Object Model \(DOM\) Level 2 Core Specification](#).

Fields inherited from class org.w3c.dom.[Node](#)

[ATTRIBUTE_NODE](#), [CDATA_SECTION_NODE](#), [COMMENT_NODE](#),
[DOCUMENT_FRAGMENT_NODE](#), [DOCUMENT_NODE](#), [DOCUMENT_TYPE_NODE](#),
[ELEMENT_NODE](#), [ENTITY_NODE](#), [ENTITY_REFERENCE_NODE](#), [NOTATION_NODE](#),
[PROCESSING_INSTRUCTION_NODE](#), [TEXT_NODE](#)

Method Summary

Text	
splitText	(int offset)
	Breaks this node into two nodes at the specified offset, keeping both in the tree as siblings.

Methods inherited from interface [org.w3c.dom.CharacterData](#)

[appendData](#), [deleteData](#), [getData](#), [getLength](#), [insertData](#), [replaceData](#), [setData](#), [substringData](#)

Methods inherited from interface [org.w3c.dom.Node](#)

[appendChild](#), [cloneNode](#), [getAttributes](#), [getChildNodes](#), [getFirstChild](#), [getLastChild](#), [getLocalName](#), [getNamespaceURI](#), [getNextSibling](#), [getNodeName](#), [getNodeType](#), [getNodeValue](#), [getOwnerDocument](#), [getParentNode](#), [getPrefix](#), [getPreviousSibling](#), [hasAttributes](#), [hasChildNodes](#), [insertBefore](#), [isSupported](#), [normalize](#), [removeChild](#), [replaceChild](#), [setNodeValue](#), [setPrefix](#)

Method Detail**splitText**

```
public Text splitText(int offset)
    throws DOMException
```

Breaks this node into two nodes at the specified `offset`, keeping both in the tree as siblings.

After being split, this node will contain all the content up to the `offset` point. A new node of the same type, which contains all the content at and after the `offset` point, is returned. If the original node had a parent node, the new node is inserted as the next sibling of the original node. When the `offset` is equal to the length of this node, the new node has no data.

Parameters:

`offset` - 16-bit unit offset at which to split, starting from 0.

Returns:

The new node, of the same type as this node.

Throws:

[DOMException](#) - INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform

Class Transformer

java.lang.Object

|

+-- **javax.xml.transform.Transformer**

public abstract class **Transformer**

extends java.lang.Object

An instance of this abstract class can transform a source tree into a result tree.

An instance of this class can be obtained with the [TransformerFactory.newTransformer](#) method. This instance may then be used to process XML from a variety of sources and write the transformation output to a variety of sinks.

An object of this class may not be used in multiple threads running concurrently. Different Transformers may be used concurrently by different threads.

A Transformer may be used multiple times. Parameters and output properties are preserved across transformations.

Constructor Summary

protected	Transformer () Default constructor is protected on purpose.
-----------	---

Method Summary

abstract void	clearParameters () Clear all parameters set with setParameter.
---------------	--

abstract EventListener	getErrorListener () Get the error event handler in effect for the transformation.
--	---

abstract java.util.Properties	getOutputProperties () Get a copy of the output properties for the transformation.
-------------------------------	--

abstract java.lang.String	getOutputProperty (java.lang.String name) Get an output property that is in effect for the transformation.
---------------------------	---

abstract java.lang.Object	getParameter (java.lang.String name) Get a parameter that was explicitly set with setParameter or setParameters.
---------------------------	---

abstract URIResolver	getURIResolver () Get an object that will be used to resolve URIs used in document(), etc.
--------------------------------------	--

abstract void	<code>setErrorListener</code> (<code>ErrorListener</code> listener) Set the error event listener in effect for the transformation.
abstract void	<code>setOutputProperties</code> (java.util.Properties oformat) Set the output properties for the transformation.
abstract void	<code>setOutputProperty</code> (java.lang.String name, java.lang.String value) Set an output property that will be in effect for the transformation.
abstract void	<code>setParameter</code> (java.lang.String name, java.lang.Object value) Add a parameter for the transformation.
abstract void	<code>setURIResolver</code> (<code>URIResolver</code> resolver) Set an object that will be used to resolve URIs used in document().
abstract void	<code>transform</code> (<code>Source</code> xmlSource, <code>Result</code> outputTarget) Process the source tree to the output result.

Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`,
`toString`, `wait`, `wait`, `wait`

Constructor Detail

Transformer

protected **Transformer**()

Default constructor is protected on purpose.

Method Detail

transform

```
public abstract void transform(Source xmlSource,  
                                Result outputTarget)  
                                throws TransformerException
```

Process the source tree to the output result.

Parameters:

xmlSource - The input for the source tree.

outputTarget - The output target.

Throws:

[`TransformerException`](#) - If an unrecoverable error occurs during the course of the transformation.

setParameter

```
public abstract void setParameter(java.lang.String name,
                                   java.lang.Object value)
```

Add a parameter for the transformation.

Pass a qualified name as a two-part string, the namespace URI enclosed in curly braces ({}), followed by the local name. If the name has a null URL, the String only contain the local name. An application can safely check for a non-null URI by testing to see if the first character of the name is a '{' character.

For example, if a URI and local name were obtained from an element defined with `<xyz:foo xmlns:xyz="http://xyz.foo.com/yada/baz.html"/>`, then the qualified name would be `"{http://xyz.foo.com/yada/baz.html}foo"`. Note that no prefix is used.

Parameters:

`name` - The name of the parameter, which may begin with a namespace URI in curly braces ({}).

`value` - The value object. This can be any valid Java object. It is up to the processor to provide the proper object coercion or to simply pass the object on for use in an extension.

getParameter

```
public abstract java.lang.Object getParameter(java.lang.String name)
```

Get a parameter that was explicitly set with `setParameter` or `setParameters`.

This method does not return a default parameter value, which cannot be determined until the node context is evaluated during the transformation process.

Returns:

A parameter that has been set with `setParameter`.

clearParameters

```
public abstract void clearParameters()
```

Clear all parameters set with `setParameter`.

setURIResolver

```
public abstract void setURIResolver(URIResolver resolver)
```

Set an object that will be used to resolve URIs used in `document()`.

If the resolver argument is null, the `URIResolver` value will be cleared, and the default behavior will be used.

Parameters:

`resolver` - An object that implements the `URIResolver` interface, or null.

getURIResolver

```
public abstract URIResolver getURIResolver()
```

Get an object that will be used to resolve URIs used in document(), etc.

Returns:

An object that implements the URIResolver interface, or null.

setOutputProperties

```
public abstract void setOutputProperties(java.util.Properties oformat)
                                throws java.lang.IllegalArgumentException
```

Set the output properties for the transformation. These properties will override properties set in the Templates with xsl:output.

If argument to this function is null, any properties previously set are removed, and the value will revert to the value defined in the templates object.

Pass a qualified property key name as a two-part string, the namespace URI enclosed in curly braces ({ }), followed by the local name. If the name has a null URL, the String only contain the local name. An application can safely check for a non-null URI by testing to see if the first character of the name is a '{' character.

For example, if a URI and local name were obtained from an element defined with `<xyz:foo xmlns:xyz="http://xyz.foo.com/yada/baz.html"/>`, then the qualified name would be "{http://xyz.foo.com/yada/baz.html}foo". Note that no prefix is used.

Parameters:

of format - A set of output properties that will be used to override any of the same properties in affect for the transformation.

Throws:

java.lang.IllegalArgumentException - if any of the argument keys are not recognized and are not namespace qualified.

See Also:

[OutputKeys](#), [Properties](#)

getOutputProperties

```
public abstract java.util.Properties getOutputProperties()
```

Get a copy of the output properties for the transformation.

The properties returned should contain properties set by the user, and properties set by the stylesheet, and these properties are "defaulted" by default properties specified by [section 16 of the XSL Transformations \(XSLT\) W3C Recommendation](#). The properties that were specifically set by the user or the stylesheet should be in the base Properties list, while the XSLT default properties that were not specifically set should be the default Properties list. Thus, getOutputProperties().getProperty(String key) will obtain any property in that was set by [setOutputProperty\(java.lang.String, java.lang.String\)](#),

[setOutputProperties\(java.util.Properties\)](#), in the stylesheet, or the default properties, while `getOutputProperties().get(String key)` will only retrieve properties that were explicitly set by [setOutputProperty\(java.lang.String, java.lang.String\)](#), [setOutputProperties\(java.util.Properties\)](#), or in the stylesheet.

Note that mutation of the Properties object returned will not effect the properties that the transformation contains.

If any of the argument keys are not recognized and are not namespace qualified, the property will be ignored. In other words the behaviour is not orthogonal with `setOutputProperties`.

See Also:

[OutputKeys](#), `Properties`

setOutputProperty

```
public abstract void setOutputProperty(java.lang.String name,
                                         java.lang.String value)
                                         throws java.lang.IllegalArgumentException
```

Set an output property that will be in effect for the transformation.

Pass a qualified property name as a two-part string, the namespace URI enclosed in curly braces (`{}`), followed by the local name. If the name has a null URL, the String only contain the local name. An application can safely check for a non-null URI by testing to see if the first character of the name is a `'{'` character.

For example, if a URI and local name were obtained from an element defined with `<xyz:foo xmlns:xyz="http://xyz.foo.com/yada/baz.html"/>`, then the qualified name would be `"{http://xyz.foo.com/yada/baz.html}foo"`. Note that no prefix is used.

The Properties object that was passed to [setOutputProperties\(java.util.Properties\)](#) won't be effected by calling this method.

Parameters:

name - A non-null String that specifies an output property name, which may be namespace qualified.

value - The non-null string value of the output property.

Throws:

`java.lang.IllegalArgumentException` - If the property is not supported, and is not qualified with a namespace.

See Also:

[OutputKeys](#)

getOutputProperty

```
public abstract java.lang.String getOutputProperty(java.lang.String name)
                                         throws java.lang.IllegalArgumentException
```

Get an output property that is in effect for the transformation. The property

specified may be a property that was set with `setOutputProperty`, or it may be a property specified in the stylesheet.

Parameters:

name - A non-null String that specifies an output property name, which may be namespace qualified.

Returns:

The string value of the output property, or null if no property was found.

Throws:

`java.lang.IllegalArgumentException` - If the property is not supported.

See Also:

[OutputKeys](#)

setErrorListener

```
public abstract void setErrorListener(ErrorListener listener)
                                throws java.lang.IllegalArgumentException
```

Set the error event listener in effect for the transformation.

Parameters:

listener - The new error listener.

Throws:

`java.lang.IllegalArgumentException` - if listener is null.

getErrorListener

```
public abstract ErrorListener getErrorListener()
```

Get the error event handler in effect for the transformation.

Returns:

The current error handler, which should never be null.

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)
[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform

Class TransformerConfigurationException

java.lang.Object

|

+-- java.lang.Throwable

|

+-- java.lang.Exception

|

+-- [javax.xml.transform.TransformerException](#)

|

+-- **javax.xml.transform.TransformerConfigurationException**public class **TransformerConfigurationException**extends [TransformerException](#)

Indicates a serious configuration error.

See Also:[Serialized Form](#)

Constructor Summary

[TransformerConfigurationException](#)()

Create a new TransformerConfigurationException with no detail message.

[TransformerConfigurationException](#)(java.lang.String msg)

Create a new TransformerConfigurationException with the String specified as an error message.

[TransformerConfigurationException](#)(java.lang.String message, [SourceLocator](#) locator)

Create a new TransformerConfigurationException from a message and a Locator.

[TransformerConfigurationException](#)(java.lang.String message, [SourceLocator](#) locator, java.lang.Throwable e)

Wrap an existing exception in a TransformerConfigurationException.

[TransformerConfigurationException](#)(java.lang.String msg, java.lang.Throwable e)

Create a new TransformerConfigurationException with the given Exception base cause and detail message.

[**TransformerConfigurationException**](#)(java.lang.Throwable e)

Create a new TransformerConfigurationException with a given Exception base cause of the error.

Methods inherited from class javax.xml.transform.[**TransformerException**](#)

[getCause](#), [getException](#), [getLocationAsString](#), [getLocator](#),
[getMessageAndLocation](#), [initCause](#), [printStackTrace](#), [printStackTrace](#),
[printStackTrace](#), [setLocator](#)

Methods inherited from class java.lang.Throwable

[fillInStackTrace](#), [getLocalizedMessage](#), [getMessage](#), [toString](#)

Methods inherited from class java.lang.Object

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [wait](#),
[wait](#), [wait](#)

Constructor Detail

TransformerConfigurationException

```
public TransformerConfigurationException()
```

Create a new TransformerConfigurationException with no detail message.

TransformerConfigurationException

```
public TransformerConfigurationException( java.lang.String msg)
```

Create a new TransformerConfigurationException with the String specified as an error message.

Parameters:

msg - The error message for the exception.

TransformerConfigurationException

```
public TransformerConfigurationException( java.lang.Throwable e)
```

Create a new TransformerConfigurationException with a given Exception base cause of the error.

Parameters:

e - The exception to be encapsulated in a TransformerConfigurationException.

TransformerConfigurationException

```
public TransformerConfigurationException( java.lang.String msg,
                                           java.lang.Throwable e )
```

Create a new TransformerConfigurationException with the given Exception base cause and detail message.

Parameters:

e - The exception to be encapsulated in a TransformerConfigurationException

msg - The detail message.

e - The exception to be wrapped in a TransformerConfigurationException

TransformerConfigurationException

```
public TransformerConfigurationException( java.lang.String message,
                                           SourceLocator locator )
```

Create a new TransformerConfigurationException from a message and a Locator.

This constructor is especially useful when an application is creating its own exception from within a DocumentHandler callback.

Parameters:

message - The error or warning message.

locator - The locator object for the error or warning.

TransformerConfigurationException

```
public TransformerConfigurationException( java.lang.String message,
                                           SourceLocator locator,
                                           java.lang.Throwable e )
```

Wrap an existing exception in a TransformerConfigurationException.

Parameters:

message - The error or warning message, or null to use the message from the embedded exception.

locator - The locator object for the error or warning.

e - Any exception.

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform

Class TransformerException

java.lang.Object

|

+-- java.lang.Throwable

|

+-- java.lang.Exception

|

+-- **javax.xml.transform.TransformerException**

Direct Known Subclasses:

[TransformerConfigurationException](#)
public class **TransformerException**

extends java.lang.Exception

This class specifies an exceptional condition that occurred during the transformation process.

See Also:

[Serialized Form](#)

Constructor Summary

[TransformerException](#)(java.lang.String message)

Create a new TransformerException.

[TransformerException](#)(java.lang.String message, [SourceLocator](#) locator)

Create a new TransformerException from a message and a Locator.

[TransformerException](#)(java.lang.String message, [SourceLocator](#) locator, java.lang.Throwable e)

Wrap an existing exception in a TransformerException.

[TransformerException](#)(java.lang.String message, java.lang.Throwable e)

Wrap an existing exception in a TransformerException.

[TransformerException](#)(java.lang.Throwable e)

Create a new TransformerException wrapping an existing exception.

Method Summary

java.lang.Throwable	getCause () Returns the cause of this throwable or null if the cause is nonexistent or unknown.
java.lang.Throwable	getException () This method retrieves an exception that this exception wraps.
java.lang.String	getLocationAsString () Get the location information as a string.
SourceLocator	getLocator () Method getLocator retrieves an instance of a SourceLocator object that specifies where an error occurred.
java.lang.String	getMessageAndLocation () Get the error message with location information appended.
java.lang.Throwable	initCause (java.lang.Throwable cause) Initializes the <i>cause</i> of this throwable to the specified value.
void	printStackTrace () Print the the trace of methods from where the error originated.
void	printStackTrace (java.io.PrintStream s) Print the the trace of methods from where the error originated.
void	printStackTrace (java.io.PrintWriter s) Print the the trace of methods from where the error originated.
void	setLocator (SourceLocator location) Method setLocator sets an instance of a SourceLocator object that specifies where an error occurred.

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

TransformerException

```
public TransformerException( java.lang.String message)
```

Create a new TransformerException.

Parameters:

message - The error or warning message.

TransformerException

```
public TransformerException( java.lang.Throwable e)
```

Create a new TransformerException wrapping an existing exception.

Parameters:

e - The exception to be wrapped.

TransformerException

```
public TransformerException( java.lang.String message,  
                             java.lang.Throwable e)
```

Wrap an existing exception in a TransformerException.

This is used for throwing processor exceptions before the processing has started.

Parameters:

message - The error or warning message, or null to use the message from the embedded exception.

e - Any exception

TransformerException

```
public TransformerException( java.lang.String message,  
                             SourceLocator locator)
```

Create a new TransformerException from a message and a Locator.

This constructor is especially useful when an application is creating its own exception from within a DocumentHandler callback.

Parameters:

message - The error or warning message.

`locator` - The locator object for the error or warning.

TransformerException

```
public TransformerException( java.lang.String message,  
                             SourceLocator locator,  
                             java.lang.Throwable e)
```

Wrap an existing exception in a TransformerException.

Parameters:

`message` - The error or warning message, or null to use the message from the embedded exception.

`locator` - The locator object for the error or warning.

`e` - Any exception

Method Detail

getLocator

```
public SourceLocator getLocator()
```

Method `getLocator` retrieves an instance of a `SourceLocator` object that specifies where an error occurred.

Returns:

A `SourceLocator` object, or null if none was specified.

setLocator

```
public void setLocator(SourceLocator location)
```

Method `setLocator` sets an instance of a `SourceLocator` object that specifies where an error occurred.

Parameters:

`location` - A `SourceLocator` object, or null to clear the location.

getException

```
public java.lang.Throwable getException()
```

This method retrieves an exception that this exception wraps.

Returns:

An Throwable object, or null.

See Also:

[getCause\(\)](#)

getCause

```
public java.lang.Throwable getCause()
```

Returns the cause of this throwable or null if the cause is nonexistent or unknown. (The cause is the throwable that caused this throwable to get thrown.)

initCause

```
public java.lang.Throwable initCause(java.lang.Throwable cause)
```

Initializes the *cause* of this throwable to the specified value. (The cause is the throwable that caused this throwable to get thrown.)

This method can be called at most once. It is generally called from within the constructor, or immediately after creating the throwable. If this throwable was created with [TransformerException\(Throwable\)](#) or [TransformerException\(String,Throwable\)](#), this method cannot be called even once.

Parameters:

cause - the cause (which is saved for later retrieval by the [getCause\(\)](#) method). (A null value is permitted, and indicates that the cause is nonexistent or unknown.)

Returns:

a reference to this Throwable instance.

Throws:

java.lang.IllegalArgumentException - if *cause* is this throwable. (A throwable cannot be its own cause.)

IllegalStateException - if this throwable was created with

[TransformerException\(Throwable\)](#) or

[TransformerException\(String,Throwable\)](#), or this method has already been called on this throwable.

getMessageAndLocation

```
public java.lang.String getMessageAndLocation()
```

Get the error message with location information appended.

Returns:

A `String` representing the error message with location information appended.

getLocationAsString

```
public java.lang.String getLocationAsString()
```

Get the location information as a string.

Returns:

A string with location info, or null if there is no location information.

printStackTrace

```
public void printStackTrace()
```

Print the the trace of methods from where the error originated. This will trace all nested exception objects, as well as this object.

Overrides:

`printStackTrace` in class `java.lang.Throwable`

printStackTrace

```
public void printStackTrace(java.io.PrintStream s)
```

Print the the trace of methods from where the error originated. This will trace all nested exception objects, as well as this object.

Parameters:

`s` - The stream where the dump will be sent to.

Overrides:

`printStackTrace` in class `java.lang.Throwable`

printStackTrace

```
public void printStackTrace( java.io.PrintWriter s )
```

Print the the trace of methods from where the error originated. This will trace all nested exception objects, as well as this object.

Parameters:

s - The writer where the dump will be sent to.

Overrides:

printStackTrace in class java.lang.Throwable

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform

Class TransformerFactory

java.lang.Object

|

+-- **javax.xml.transform.TransformerFactory****Direct Known Subclasses:**[SAXTransformerFactory](#)public abstract class **TransformerFactory**

extends java.lang.Object

A TransformerFactory instance can be used to create [Transformer](#) and [Templates](#) objects.

The system property that determines which Factory implementation to create is named "javax.xml.transform.TransformerFactory". This property names a concrete subclass of the TransformerFactory abstract class. If the property is not defined, a platform default is be used.

An implementation of the TransformerFactory class is *NOT* guaranteed to be thread safe. It is up to the user application to make sure about the use of the TransformerFactory from more than one thread. Alternatively the application can have one instance of the TransformerFactory per thread. An application can use the same instance of the factory to obtain one or more instances of a Transformer or Templates provided the instance of the factory isn't being used in more than one thread at a time.

Constructor Summary

protected	TransformerFactory () Default constructor is protected on purpose.
-----------	--

Method Summary

abstract Source	getAssociatedStylesheet (Source source, java.lang.String media, java.lang.String title, java.lang.String charset) Get the stylesheet specification(s) associated via the xml-stylesheet processing instruction (see http://www.w3.org/TR/xml-stylesheet/) with the document document specified in the source parameter, and that match the given criteria.
abstract java.lang.Object	getAttribute (java.lang.String name) Allows the user to retrieve specific attributes on the underlying implementation.

abstract EventListener	getErrorListener () Get the error event handler for the TransformerFactory.
abstract boolean	getFeature (java.lang.String name) Look up the value of a feature.
abstract URIResolver	getURIResolver () Get the object that is used by default during the transformation to resolve URIs used in document(), xsl:import, or xsl:include.
static TransformerFactory	newInstance () Obtain a new instance of a TransformerFactory.
abstract Templates	newTemplates (Source source) Process the Source into a Templates object, which is a compiled representation of the source.
abstract Transformer	newTransformer () Create a new Transformer object that performs a copy of the source to the result.
abstract Transformer	newTransformer (Source source) Process the Source into a Transformer object.
abstract void	setAttribute (java.lang.String name , java.lang.Object value) Allows the user to set specific attributes on the underlying implementation.
abstract void	setErrorListener (EventListener listener) Set the error event listener for the TransformerFactory, which is used for the processing of transformation instructions, and not for the transformation itself.
abstract void	setURIResolver (URIResolver resolver) Set an object that is used by default during the transformation to resolve URIs used in xsl:import, or xsl:include.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TransformerFactory

protected **TransformerFactory**()

Default constructor is protected on purpose.

Method Detail

newInstance

```
public static TransformerFactory newInstance()
                                   throws TransformerFactoryConfigurationError
```

Obtain a new instance of a `TransformerFactory`. This static method creates a new factory instance. This method uses the following ordered lookup procedure to determine the `TransformerFactory` implementation class to load:

- Use the `javax.xml.transform.TransformerFactory` system property.
- Use the properties file "lib/jaxp.properties" in the JRE directory. This configuration file is in standard `java.util.Properties` format and contains the fully qualified name of the implementation class with the key being the system property defined above.
- Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file `META-INF/services/javax.xml.transform.TransformerFactory` in jars available to the runtime.
- Platform default `TransformerFactory` instance.

Once an application has obtained a reference to a `TransformerFactory` it can use the factory to configure and obtain parser instances.

Returns:

new `TransformerFactory` instance, never null.

Throws:

[TransformerFactoryConfigurationError](#) - if the implementation is not available or cannot be instantiated.

newTransformer

```
public abstract Transformer newTransformer(Source source)
                                   throws TransformerConfigurationException
```

Process the `Source` into a `Transformer` object. Care must be given not to use this object in multiple threads running concurrently. Different `TransformerFactories` can be used concurrently by different threads.

Parameters:

`source` - An object that holds a URI, input stream, etc.

Returns:

A `Transformer` object that may be used to perform a transformation in a single thread, never null.

Throws:

[TransformerConfigurationException](#) - May throw this during the parse when it is constructing the Templates object and fails.

newTransformer

```
public abstract Transformer newTransformer()
                                   throws TransformerConfigurationException
```

Create a new `Transformer` object that performs a copy of the source to the result.

Parameters:

source - An object that holds a URI, input stream, etc.

Returns:

A Transformer object that may be used to perform a transformation in a single thread, never null.

Throws:

[TransformerConfigurationException](#) - May throw this during the parse when it is constructing the Templates object and fails.

newTemplates

```
public abstract Templates newTemplates(Source source)
                                   throws TransformerConfigurationException
```

Process the Source into a Templates object, which is a compiled representation of the source. This Templates object may then be used concurrently across multiple threads. Creating a Templates object allows the TransformerFactory to do detailed performance optimization of transformation instructions, without penalizing runtime transformation.

Parameters:

source - An object that holds a URL, input stream, etc.

Returns:

A Templates object capable of being used for transformation purposes, never null.

Throws:

[TransformerConfigurationException](#) - May throw this during the parse when it is constructing the Templates object and fails.

getAssociatedStylesheet

```
public abstract Source getAssociatedStylesheet(Source source,
                                                java.lang.String media,
                                                java.lang.String title,
                                                java.lang.String charset)
                                   throws TransformerConfigurationException
```

Get the stylesheet specification(s) associated via the xml-stylesheet processing instruction (see <http://www.w3.org/TR/xml-stylesheet/>) with the document document specified in the source parameter, and that match the given criteria. Note that it is possible to return several stylesheets, in which case they are applied as if they were a list of imports or cascades in a single stylesheet.

Parameters:

source - The XML source document.

media - The media attribute to be matched. May be null, in which case the preferred templates will be used (i.e. alternate = no).

title - The value of the title attribute to match. May be null.

charset - The value of the charset attribute to match. May be null.

Returns:

A Source object suitable for passing to the TransformerFactory.

Throws:

TransformerConfigurationException. -

setURIResolver

```
public abstract void setURIResolver(URIResolver resolver)
```

Set an object that is used by default during the transformation to resolve URIs used in `xsl:import`, or `xsl:include`.

Parameters:

`resolver` - An object that implements the `URIResolver` interface, or null.

getURIResolver

```
public abstract URIResolver getURIResolver()
```

Get the object that is used by default during the transformation to resolve URIs used in `document()`, `xsl:import`, or `xsl:include`.

Returns:

The `URIResolver` that was set with `setURIResolver`.

getFeature

```
public abstract boolean getFeature(java.lang.String name)
```

Look up the value of a feature.

The feature name is any absolute URI.

Parameters:

`name` - The feature name, which is an absolute URI.

Returns:

The current state of the feature (true or false).

setAttribute

```
public abstract void setAttribute(java.lang.String name,  
                                   java.lang.Object value)  
    throws java.lang.IllegalArgumentException
```

Allows the user to set specific attributes on the underlying implementation. An attribute in this context is defined to be an option that the implementation provides.

Parameters:

`name` - The name of the attribute.

`value` - The value of the attribute.

Throws:

`java.lang.IllegalArgumentException` - thrown if the underlying implementation doesn't recognize the attribute.

getAttribute

```
public abstract java.lang.Object getAttribute(java.lang.String name)
                                   throws java.lang.IllegalArgumentException
```

Allows the user to retrieve specific attributes on the underlying implementation.

Parameters:

name - The name of the attribute.

Returns:

value The value of the attribute.

Throws:

`java.lang.IllegalArgumentException` - thrown if the underlying implementation doesn't recognize the attribute.

setErrorListener

```
public abstract void setErrorListener(ErrorListener listener)
                                   throws java.lang.IllegalArgumentException
```

Set the error event listener for the TransformerFactory, which is used for the processing of transformation instructions, and not for the transformation itself.

Parameters:

listener - The new error listener.

Throws:

`java.lang.IllegalArgumentException` - if listener is null.

getErrorListener

```
public abstract ErrorListener getErrorListener()
```

Get the error event handler for the TransformerFactory.

Returns:

The current error handler, which should never be null.

[Overview](#) [Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform

Class TransformerFactoryConfigurationError

java.lang.Object

|

+-- java.lang.Throwable

|

+-- java.lang.Error

|

+-- javax.xml.transform.TransformerFactoryConfigurationError

```
public class TransformerFactoryConfigurationError
```

```
extends java.lang.Error
```

Thrown when a problem with configuration with the Transformer Factories exists. This error will typically be thrown when the class of a transformation factory specified in the system properties cannot be found or instantiated.

See Also:

[Serialized Form](#)

Constructor Summary

[TransformerFactoryConfigurationError](#)()

Create a new TransformerFactoryConfigurationError with no detail message.

[TransformerFactoryConfigurationError](#)(java.lang.Exception e)

Create a new TransformerFactoryConfigurationError with a given Exception base cause of the error.

[TransformerFactoryConfigurationError](#)(java.lang.Exception e, java.lang.String msg)

Create a new TransformerFactoryConfigurationError with the given Exception base cause and detail message.

[TransformerFactoryConfigurationError](#)(java.lang.String msg)

Create a new TransformerFactoryConfigurationError with the String specified as an error message.

Method Summary

<code>java.lang.Exception</code>	<code>getException</code> () Return the actual exception (if any) that caused this exception to be raised.
<code>java.lang.String</code>	<code>getMessage</code> () Return the message (if any) for this error .

Methods inherited from class `java.lang.Throwable`

`fillInStackTrace`, `getLocalizedMessage`, `printStackTrace`,
`printStackTrace`, `printStackTrace`, `toString`

Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`,
`wait`, `wait`

Constructor Detail

TransformerFactoryConfigurationError

```
public TransformerFactoryConfigurationError( )
```

Create a new TransformerFactoryConfigurationError with no detail mesage.

TransformerFactoryConfigurationError

```
public TransformerFactoryConfigurationError( java.lang.String msg)
```

Create a new TransformerFactoryConfigurationError with the String specified as an error message.

Parameters:

`msg` - The error message for the exception.

TransformerFactoryConfigurationError

```
public TransformerFactoryConfigurationError( java.lang.Exception e)
```

Create a new TransformerFactoryConfigurationError with a given Exception base cause of the error.

Parameters:

e - The exception to be encapsulated in a TransformerFactoryConfigurationError.

TransformerFactoryConfigurationError

```
public TransformerFactoryConfigurationError( java.lang.Exception e,  
                                             java.lang.String msg)
```

Create a new TransformerFactoryConfigurationError with the given Exception base cause and detail message.

Parameters:

e - The exception to be encapsulated in a TransformerFactoryConfigurationError

msg - The detail message.

e - The exception to be wrapped in a TransformerFactoryConfigurationError

Method Detail

getMessage

```
public java.lang.String getMessage()
```

Return the message (if any) for this error . If there is no message for the exception and there is an encapsulated exception then the message of that exception will be returned.

Returns:

The error message.

Overrides:

getMessage in class java.lang.Throwable

getException

```
public java.lang.Exception getException()
```

Return the actual exception (if any) that caused this exception to be raised.

Returns:

The encapsulated exception, or null if there is none.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform.sax

Interface TransformerHandler

public abstract interface **TransformerHandler**extends [ContentHandler](#), [LexicalHandler](#), [DTDHandler](#)

A TransformerHandler listens for SAX ContentHandler parse events and transforms them to a Result.

Method Summary

java.lang.String	getSystemId () Get the base ID (URI or system ID) from where relative URLs will be resolved.
Transformer	getTransformer () Get the Transformer associated with this handler, which is needed in order to set parameters and output properties.
void	setResult (Result result) Enables the user of the TransformerHandler to set the to set the Result for the transformation.
void	setSystemId (java.lang.String systemID) Set the base ID (URI or system ID) from where relative URLs will be resolved.

Methods inherited from interface org.xml.sax.[ContentHandler](#)

[characters](#), [endDocument](#), [endElement](#), [endPrefixMapping](#),
[ignorableWhitespace](#), [processingInstruction](#), [setDocumentLocator](#),
[skippedEntity](#), [startDocument](#), [startElement](#), [startPrefixMapping](#)

Methods inherited from interface org.xml.sax.ext.[LexicalHandler](#)

[comment](#), [endCDATA](#), [endDTD](#), [endEntity](#), [startCDATA](#), [startDTD](#),
[startEntity](#)

Methods inherited from interface [org.xml.sax.DTDHandler](#)[notationDecl](#), [unparsedEntityDecl](#)

Method Detail

setResult

```
public void setResult(Result result)
```

```
throws java.lang.IllegalArgumentException
```

Enables the user of the TransformerHandler to set the to set the Result for the transformation.

Parameters:

result - A Result instance, should not be null.

Throws:

java.lang.IllegalArgumentException - if result is invalid for some reason.

setSystemId

```
public void setSystemId(java.lang.String systemID)
```

Set the base ID (URI or system ID) from where relative URLs will be resolved.

Parameters:

systemID - Base URI for the source tree.

getSystemId

```
public java.lang.String getSystemId()
```

Get the base ID (URI or system ID) from where relative URLs will be resolved.

Returns:

The systemID that was set with [setSystemId\(java.lang.String\)](#).

getTransformer

```
public Transformer getTransformer()
```


Get the Transformer associated with this handler, which is needed in order to set parameters and output properties.

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.transform

Interface URIResolver

public abstract interface **URIResolver**

An object that implements this interface that can be called by the processor to turn a URI used in document(), xsl:import, or xsl:include into a Source object.

Method Summary

Source	resolve (java.lang.String href, java.lang.String base) Called by the processor when it encounters an xsl:include, xsl:import, or document() function.
------------------------	---

Method Detail

resolve

```
public Source resolve(java.lang.String href,  
                      java.lang.String base)  
    throws TransformerException
```

Called by the processor when it encounters an xsl:include, xsl:import, or document() function.

Parameters:

href - An href attribute, which may be relative or absolute.

base - The base URI in effect when the href attribute was encountered.

Returns:

A Source object, or null if the href cannot be resolved, and the processor should try to resolve the URI itself.

Throws:

[TransformerException](#) - if an error occurs when trying to resolve the URI.

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)

Submit Feedback to xml-feedback@java.sun.com

Copyright 1998-2001 [Sun Microsystems Inc.](#)

All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax

Interface XMLFilter

All Known Implementing Classes:

[XMLFilterImpl](#)public abstract interface **XMLFilter**extends [XMLReader](#)

Interface for an XML filter.

This module, both source code and documentation, is in the Public Domain, and comes with NO WARRANTY.

An XML filter is like an XML reader, except that it obtains its events from another XML reader rather than a primary source like an XML document or database. Filters can modify a stream of events as they pass on to the final application.

The XMLFilterImpl helper class provides a convenient base for creating SAX2 filters, by passing on all [EntityResolver](#), [DTDHandler](#), [ContentHandler](#) and [ErrorHandler](#) events automatically.

Since:

SAX 2.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[XMLFilterImpl](#)

Method Summary

[XMLReader](#)**[getParent](#)** ()

Get the parent reader.

void

[setParent](#) ([XMLReader](#) parent)

Set the parent reader.

Methods inherited from interface [org.xml.sax.XMLReader](#)

[getContentHandler](#), [getDTDHandler](#), [getEntityResolver](#), [getErrorHandler](#), [getFeature](#), [getProperty](#), [parse](#), [parse](#), [setContentHandler](#), [setDTDHandler](#), [setEntityResolver](#), [setErrorHandler](#), [setFeature](#), [setProperty](#)

Method Detail**setParent**

```
public void setParent(XMLReader parent)
```

Set the parent reader.

This method allows the application to link the filter to a parent reader (which may be another filter). The argument may not be null.

Parameters:

parent - The parent reader.

getParent

```
public XMLReader getParent()
```

Get the parent reader.

This method allows the application to query the parent reader (which may be another filter). It is generally a bad idea to perform any operations on the parent reader directly: they should all pass through this filter.

Returns:

The parent filter, or null if none has been set.

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax.helpers

Class XMLFilterImpl

java.lang.Object

```

|
+--org.xml.sax.helpers.XMLFilterImpl

```

public class **XMLFilterImpl**

extends java.lang.Object

implements [XMLFilter](#), [EntityResolver](#), [DTDHandler](#), [ContentHandler](#), [ErrorHandler](#)

Base class for deriving an XML filter.

This module, both source code and documentation, is in the Public Domain, and comes with NO WARRANTY.

This class is designed to sit between an [XMLReader](#) and the client application's event handlers. By default, it does nothing but pass requests up to the reader and events on to the handlers unmodified, but subclasses can override specific methods to modify the event stream or the configuration requests as they pass through.

Since:

SAX 2.0

Version:

2.0

Author:

David Megginson, sax@megginson.com

See Also:

[XMLFilter](#), [XMLReader](#), [EntityResolver](#), [DTDHandler](#), [ContentHandler](#), [ErrorHandler](#)

Constructor Summary

[XMLFilterImpl](#) ()

Construct an empty XML filter, with no parent.

[XMLFilterImpl](#) ([XMLReader](#) parent)

Construct an XML filter with the specified parent.

Method Summary

void	characters (char[] ch, int start, int length) Filter a character data event.
void	endDocument () Filter an end document event.
void	endElement (java.lang.String uri, java.lang.String localName, java.lang.String qName) Filter an end element event.
void	endPrefixMapping (java.lang.String prefix) Filter an end Namespace prefix mapping event.
void	error (SAXParseException e) Filter an error event.
void	fatalError (SAXParseException e) Filter a fatal error event.
ContentHandler	getContentHandler () Get the content event handler.
DTDHandler	getDTDHandler () Get the current DTD event handler.
EntityResolver	getEntityResolver () Get the current entity resolver.
ErrorHandler	getErrorHandler () Get the current error event handler.
boolean	getFeature (java.lang.String name) Look up the state of a feature.
XMLReader	getParent () Get the parent reader.
java.lang.Object	getProperty (java.lang.String name) Look up the value of a property.
void	ignorableWhitespace (char[] ch, int start, int length) Filter an ignorable whitespace event.
void	notationDecl (java.lang.String name, java.lang.String publicId, java.lang.String systemId) Filter a notation declaration event.
void	parse (InputSource input) Parse a document.

void	parse (java.lang.String systemId) Parse a document.
void	processingInstruction (java.lang.String target, java.lang.String data) Filter a processing instruction event.
<u>InputSource</u>	resolveEntity (java.lang.String publicId, java.lang.String systemId) Filter an external entity resolution.
void	setContentHandler (<u>ContentHandler</u> handler) Set the content event handler.
void	setDocumentLocator (<u>Locator</u> locator) Filter a new document locator event.
void	setDTDHandler (<u>DTDHandler</u> handler) Set the DTD event handler.
void	setEntityResolver (<u>EntityResolver</u> resolver) Set the entity resolver.
void	setErrorHandler (<u>ErrorHandler</u> handler) Set the error event handler.
void	setFeature (java.lang.String name, boolean state) Set the state of a feature.
void	setParent (<u>XMLReader</u> parent) Set the parent reader.
void	setProperty (java.lang.String name, java.lang.Object value) Set the value of a property.
void	skippedEntity (java.lang.String name) Filter a skipped entity event.
void	startDocument () Filter a start document event.
void	startElement (java.lang.String uri, java.lang.String localName, java.lang.String qName, <u>Attributes</u> atts) Filter a start element event.
void	startPrefixMapping (java.lang.String prefix, java.lang.String uri) Filter a start Namespace prefix mapping event.

void	<code>unparsedEntityDecl</code> (java.lang.String name, java.lang.String publicId, java.lang.String systemId, java.lang.String notationName) Filter an unparsed entity declaration event.
void	<code>warning</code> (<code>SAXParseException</code> e) Filter a warning event.

Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`,
`toString`, `wait`, `wait`, `wait`

Constructor Detail

XMLFilterImpl

```
public XMLFilterImpl()
```

Construct an empty XML filter, with no parent.

This filter will have no parent: you must assign a parent before you start a parse or do any configuration with `setFeature` or `setProperty`.

See Also:

[`XMLReader.setFeature\(java.lang.String, boolean\)`](#),
[`XMLReader.setProperty\(java.lang.String, java.lang.Object\)`](#)

XMLFilterImpl

```
public XMLFilterImpl(XMLReader parent)
```

Construct an XML filter with the specified parent.

See Also:

[`setParent\(org.xml.sax.XMLReader\)`](#), [`getParent\(\)`](#)

Method Detail

setParent

```
public void setParent(XMLReader parent)
```

Set the parent reader.

This is the [XMLReader](#) from which this filter will obtain its events and to which it will pass its configuration requests. The parent may itself be another filter.

If there is no parent reader set, any attempt to parse or to set or get a feature or property will fail.

Specified by:

[setParent](#) in interface [XMLFilter](#)

Parameters:

parent - The parent XML reader.

Throws:

java.lang.NullPointerException - If the parent is null.

See Also:

[getParent\(\)](#)

getParent

```
public XMLReader getParent()
```

Get the parent reader.

Specified by:

[getParent](#) in interface [XMLFilter](#)

Returns:

The parent XML reader, or null if none is set.

See Also:

[setParent\(org.xml.sax.XMLReader\)](#)

setFeature

```
public void setFeature(java.lang.String name,  
                        boolean state)  
    throws SAXNotRecognizedException,  
           SAXNotSupportedException
```

Set the state of a feature.

This will always fail if the parent is null.

Parameters:

`name` - The feature name.

`state` - The requested feature state.

Throws:

[SAXNotRecognizedException](#) - When the XMLReader does not recognize the feature name.

[SAXNotSupportedException](#) - When the XMLReader recognizes the feature name but cannot set the requested value.

See Also:

[XMLReader.setFeature\(java.lang.String, boolean\)](#)

getFeature

```
public boolean getFeature(java.lang.String name)
    throws SAXNotRecognizedException,
           SAXNotSupportedException
```

Look up the state of a feature.

This will always fail if the parent is null.

Parameters:

`name` - The feature name.

Returns:

The current state of the feature.

Throws:

[SAXNotRecognizedException](#) - When the XMLReader does not recognize the feature name.

[SAXNotSupportedException](#) - When the XMLReader recognizes the feature name but cannot determine its state at this time.

See Also:

[XMLReader.getFeature\(java.lang.String\)](#)

setProperty

```
public void setProperty(java.lang.String name,
    java.lang.Object value)
```

throws [SAXNotRecognizedException](#),
[SAXNotSupportedException](#)

Set the value of a property.

This will always fail if the parent is null.

Parameters:

name - The property name.

state - The requested property value.

Throws:

[SAXNotRecognizedException](#) - When the XMLReader does not recognize the property name.

[SAXNotSupportedException](#) - When the XMLReader recognizes the property name but cannot set the requested value.

See Also:

[XMLReader.setProperty\(java.lang.String, java.lang.Object\)](#)

getProperty

```
public java.lang.Object getProperty(java.lang.String name)  
                                throws SAXNotRecognizedException,  
                                       SAXNotSupportedException
```

Look up the value of a property.

Parameters:

name - The property name.

Returns:

The current value of the property.

Throws:

[SAXNotRecognizedException](#) - When the XMLReader does not recognize the feature name.

[SAXNotSupportedException](#) - When the XMLReader recognizes the property name but cannot determine its value at this time.

See Also:

[XMLReader.setFeature\(java.lang.String, boolean\)](#)

setEntityResolver

```
public void setEntityResolver(EntityResolver resolver)
```

Set the entity resolver.

Parameters:

resolver - The new entity resolver.

Throws:

java.lang.NullPointerException - If the resolver is null.

See Also:

[XMLReader.setEntityResolver\(org.xml.sax.EntityResolver\)](#)

getEntityResolver

```
public EntityResolver getEntityResolver()
```

Get the current entity resolver.

Returns:

The current entity resolver, or null if none was set.

See Also:

[XMLReader.getEntityResolver\(\)](#)

setDTDHandler

```
public void setDTDHandler(DTDHandler handler)
```

Set the DTD event handler.

Parameters:

resolver - The new DTD handler.

Throws:

java.lang.NullPointerException - If the handler is null.

See Also:

[XMLReader.setDTDHandler\(org.xml.sax.DTDHandler\)](#)

getDTDHandler

```
public DTDHandler getDTDHandler()
```

Get the current DTD event handler.

Returns:

The current DTD handler, or null if none was set.

See Also:

[XMLReader.getDTDHandler\(\)](#)

setContentHandler

```
public void setContentHandler(ContentHandler handler)
```

Set the content event handler.

Parameters:

resolver - The new content handler.

Throws:

java.lang.NullPointerException - If the handler is null.

See Also:

[XMLReader.setContentHandler\(org.xml.sax.ContentHandler\)](#)

getContentHandler

```
public ContentHandler getContentHandler()
```

Get the content event handler.

Returns:

The current content handler, or null if none was set.

See Also:

[XMLReader.getContentHandler\(\)](#)

setErrorHandler

```
public void setErrorHandler(ErrorHandler handler)
```

Set the error event handler.

Parameters:

`handle` - The new error handler.

Throws:

`java.lang.NullPointerException` - If the handler is null.

See Also:

[`XMLReader.setErrorHandler\(org.xml.sax.ErrorHandler\)`](#)

getErrorHandler

```
public ErrorHandler getErrorHandler()
```

Get the current error event handler.

Returns:

The current error handler, or null if none was set.

See Also:

[`XMLReader.getErrorHandler\(\)`](#)

parse

```
public void parse(InputSource input)
    throws SAXException,
           java.io.IOException
```

Parse a document.

Parameters:

`input` - The input source for the document entity.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

`java.io.IOException` - An IO exception from the parser, possibly from a byte stream or character stream supplied by the application.

See Also:

[`XMLReader.parse\(org.xml.sax.InputSource\)`](#)

parse

```
public void parse(java.lang.String systemId)
```

throws [SAXException](#),
java.io.IOException

Parse a document.

Parameters:

systemId - The system identifier as a fully-qualified URI.

Throws:

[SAXException](#) - Any SAX exception, possibly wrapping another exception.

java.io.IOException - An IO exception from the parser, possibly from a byte stream or character stream supplied by the application.

See Also:

[XMLReader.parse\(java.lang.String\)](#)

resolveEntity

```
public InputSource resolveEntity(java.lang.String publicId,  
                                java.lang.String systemId)  
    throws SAXException,  
           java.io.IOException
```

Filter an external entity resolution.

Specified by:

[resolveEntity](#) in interface [EntityResolver](#)

Parameters:

publicId - The entity's public identifier, or null.

systemId - The entity's system identifier.

Returns:

A new InputSource or null for the default.

Throws:

[SAXException](#) - The client may throw an exception during processing.

java.io.IOException - The client may throw an I/O-related exception while obtaining the new InputSource.

See Also:

[EntityResolver.resolveEntity\(java.lang.String,
java.lang.String\)](#)

notationDecl

```
public void notationDecl(java.lang.String name,  
                           java.lang.String publicId,  
                           java.lang.String systemId)  
    throws SAXException
```

Filter a notation declaration event.

Specified by:

[notationDecl](#) in interface [DTDHandler](#)

Parameters:

name - The notation name.

publicId - The notation's public identifier, or null.

systemId - The notation's system identifier, or null.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[DTDHandler.notationDecl\(java.lang.String, java.lang.String, java.lang.String\)](#)

unparsedEntityDecl

```
public void unparsedEntityDecl(java.lang.String name,  
                                java.lang.String publicId,  
                                java.lang.String systemId,  
                                java.lang.String notationName)  
    throws SAXException
```

Filter an unparsed entity declaration event.

Specified by:

[unparsedEntityDecl](#) in interface [DTDHandler](#)

Parameters:

name - The entity name.

publicId - The entity's public identifier, or null.

systemId - The entity's system identifier, or null.

notationName - The name of the associated notation.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[DTDHandler.unparsedEntityDecl\(java.lang.String, java.lang.String, java.lang.String, java.lang.String\)](#)

setDocumentLocator

```
public void setDocumentLocator(Locator locator)
```

Filter a new document locator event.

Specified by:

[setDocumentLocator](#) in interface [ContentHandler](#)

Parameters:

locator - The document locator.

See Also:

[ContentHandler.setDocumentLocator\(org.xml.sax.Locator\)](#)

startDocument

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

Filter a start document event.

Specified by:

[startDocument](#) in interface [ContentHandler](#)

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ContentHandler.startDocument\(\)](#)

endDocument

```
public void endDocument()  
           throws SAXException
```

Filter an end document event.

Specified by:

[endDocument](#) in interface [ContentHandler](#)

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ContentHandler.endDocument\(\)](#)

startPrefixMapping

```
public void startPrefixMapping(java.lang.String prefix,  
                               java.lang.String uri)  
    throws SAXException
```

Filter a start Namespace prefix mapping event.

Specified by:

[startPrefixMapping](#) in interface [ContentHandler](#)

Parameters:

`prefix` - The Namespace prefix.

`uri` - The Namespace URI.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ContentHandler.startPrefixMapping\(java.lang.String,
java.lang.String\)](#)

endPrefixMapping

```
public void endPrefixMapping(java.lang.String prefix)  
    throws SAXException
```

Filter an end Namespace prefix mapping event.

Specified by:

[endPrefixMapping](#) in interface [ContentHandler](#)

Parameters:

`prefix` - The Namespace prefix.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ContentHandler.endPrefixMapping\(java.lang.String\)](#)

startElement

```
public void startElement(java.lang.String uri,  
                        java.lang.String localName,  
                        java.lang.String qName,  
                        Attributes atts)  
    throws SAXException
```

Filter a start element event.

Specified by:

[startElement](#) in interface [ContentHandler](#)

Parameters:

`uri` - The element's Namespace URI, or the empty string.

`localName` - The element's local name, or the empty string.

`qName` - The element's qualified (prefixed) name, or the empty string.

`atts` - The element's attributes.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ContentHandler.startElement\(java.lang.String,
java.lang.String, java.lang.String, org.xml.sax.Attributes\)](#)

endElement

```
public void endElement(java.lang.String uri,  
                      java.lang.String localName,  
                      java.lang.String qName)  
    throws SAXException
```

Filter an end element event.

Specified by:

[endElement](#) in interface [ContentHandler](#)

Parameters:

`uri` - The element's Namespace URI, or the empty string.

`localName` - The element's local name, or the empty string.

`qName` - The element's qualified (prefixed) name, or the empty string.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ContentHandler.endElement\(java.lang.String, java.lang.String, java.lang.String\)](#)

characters

```
public void characters(char[] ch,  
                      int start,  
                      int length)  
    throws SAXException
```

Filter a character data event.

Specified by:

[characters](#) in interface [ContentHandler](#)

Parameters:

ch - An array of characters.

start - The starting position in the array.

length - The number of characters to use from the array.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ContentHandler.characters\(char\[\], int, int\)](#)

ignorableWhitespace

```
public void ignorableWhitespace(char[] ch,  
                                int start,  
                                int length)  
    throws SAXException
```

Filter an ignorable whitespace event.

Specified by:

[ignorableWhitespace](#) in interface [ContentHandler](#)

Parameters:

ch - An array of characters.

start - The starting position in the array.

length - The number of characters to use from the array.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ContentHandler.ignorableWhitespace\(char\[\], int, int\)](#)

processingInstruction

```
public void processingInstruction(java.lang.String target,
                                   java.lang.String data)
                                   throws SAXException
```

Filter a processing instruction event.

Specified by:

[processingInstruction](#) in interface [ContentHandler](#)

Parameters:

target - The processing instruction target.

data - The text following the target.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ContentHandler.processingInstruction\(java.lang.String, java.lang.String\)](#)

skippedEntity

```
public void skippedEntity(java.lang.String name)
                           throws SAXException
```

Filter a skipped entity event.

Specified by:

[skippedEntity](#) in interface [ContentHandler](#)

Parameters:

name - The name of the skipped entity.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ContentHandler.skippedEntity\(java.lang.String\)](#)

warning

```
public void warning(SAXParseException e)
    throws SAXException
```

Filter a warning event.

Specified by:

[warning](#) in interface [ErrorHandler](#)

Parameters:

e - The warning as an exception.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ErrorHandler.warning\(org.xml.sax.SAXParseException\)](#)

error

```
public void error(SAXParseException e)
    throws SAXException
```

Filter an error event.

Specified by:

[error](#) in interface [ErrorHandler](#)

Parameters:

e - The error as an exception.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ErrorHandler.error\(org.xml.sax.SAXParseException\)](#)

fatalError

```
public void fatalError(SAXParseException e)
    throws SAXException
```

Filter a fatal error event.

Specified by:

[fatalError](#) in interface [ErrorHandler](#)

Parameters:

e - The error as an exception.

Throws:

[SAXException](#) - The client may throw an exception during processing.

See Also:

[ErrorHandler.fatalError\(org.xml.sax.SAXParseException\)](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xml.sax.helpers

Class XMLReaderAdapter

java.lang.Object

```

|
+--org.xml.sax.helpers.XMLReaderAdapter

```

public class **XMLReaderAdapter**

extends java.lang.Object

implements [Parser](#), [ContentHandler](#)

Adapt a SAX2 XMLReader as a SAX1 Parser.

This module, both source code and documentation, is in the Public Domain, and comes with NO WARRANTY.

This class wraps a SAX2 [XMLReader](#) and makes it act as a SAX1 [Parser](#). The XMLReader must support a true value for the <http://xml.org/sax/features/namespace-prefixes> property or parsing will fail with a [SAXException](#); if the XMLReader supports a false value for the <http://xml.org/sax/features/namespace-prefixes> property, that will also be used to improve efficiency.

Since:

SAX 2.0

Version:

2.0

Author:David Megginson, sax@megginson.com**See Also:**[Parser](#), [XMLReader](#)

Constructor Summary

[XMLReaderAdapter](#) ()

Create a new adapter.

[XMLReaderAdapter](#) ([XMLReader](#) xmlReader)

Create a new adapter.

Method Summary

void	characters (char[] ch, int start, int length) Adapt a SAX2 characters event.
void	endDocument () End document event.
void	endElement (java.lang.String uri, java.lang.String localName, java.lang.String qName) Adapt a SAX2 end element event.
void	endPrefixMapping (java.lang.String prefix) Adapt a SAX2 end prefix mapping event.
void	ignorableWhitespace (char[] ch, int start, int length) Adapt a SAX2 ignorable whitespace event.
void	parse (InputSource input) Parse the document.
void	parse (java.lang.String systemId) Parse the document.
void	processingInstruction (java.lang.String target, java.lang.String data) Adapt a SAX2 processing instruction event.
void	setDocumentHandler (DocumentHandler handler) Register the SAX1 document event handler.
void	setDocumentLocator (Locator locator) Set a document locator.
void	setDTDHandler (DTDHandler handler) Register the DTD event handler.
void	setEntityResolver (EntityResolver resolver) Register the entity resolver.
void	setErrorHandler (ErrorHandler handler) Register the error event handler.
void	setLocale (java.util.Locale locale) Set the locale for error reporting.
void	skippedEntity (java.lang.String name) Adapt a SAX2 skipped entity event.
void	startDocument () Start document event.

void	startElement (java.lang.String uri, java.lang.String localName, java.lang.String qName, Attributes atts) Adapt a SAX2 start element event.
void	startPrefixMapping (java.lang.String prefix, java.lang.String uri) Adapt a SAX2 start prefix mapping event.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

XMLReaderAdapter

```
public XMLReaderAdapter()  
    throws SAXException
```

Create a new adapter.

Use the "org.xml.sax.driver" property to locate the SAX2 driver to embed.

Throws:

[SAXException](#) - If the embedded driver cannot be instantiated or if the org.xml.sax.driver property is not specified.

XMLReaderAdapter

```
public XMLReaderAdapter(XMLReader xmlReader)
```

Create a new adapter.

Create a new adapter, wrapped around a SAX2 XMLReader. The adapter will make the XMLReader act like a SAX1 Parser.

Parameters:

xmlReader - The SAX2 XMLReader to wrap.

Throws:

java.lang.NullPointerException - If the argument is null.

Method Detail

setLocale

```
public void setLocale(java.util.Locale locale)  
    throws SAXException
```

Set the locale for error reporting.

This is not supported in SAX2, and will always throw an exception.

Specified by:

[setLocale](#) in interface [Parser](#)

Parameters:

The - locale for error reporting.

See Also:

[Parser.setLocale\(java.util.Locale\)](#)

setEntityResolver

```
public void setEntityResolver(EntityResolver resolver)
```

Register the entity resolver.

Specified by:

[setEntityResolver](#) in interface [Parser](#)

Parameters:

resolver - The new resolver.

See Also:

[Parser.setEntityResolver\(org.xml.sax.EntityResolver\)](#)

setDTDHandler

```
public void setDTDHandler(DTDHandler handler)
```

Register the DTD event handler.

Specified by:

[setDTDHandler](#) in interface [Parser](#)

Parameters:

handler - The new DTD event handler.

See Also:

[Parser.setDTDHandler\(org.xml.sax.DTDHandler\)](#)

setDocumentHandler

```
public void setDocumentHandler(DocumentHandler handler)
```

Register the SAX1 document event handler.

Note that the SAX1 document handler has no Namespace support.

Specified by:

[setDocumentHandler](#) in interface [Parser](#)

Parameters:

handler - The new SAX1 document event handler.

See Also:

[Parser.setDocumentHandler\(org.xml.sax.DocumentHandler\)](#)

setErrorHandler

```
public void setErrorHandler(ErrorHandler handler)
```

Register the error event handler.

Specified by:

[setErrorHandler](#) in interface [Parser](#)

Parameters:

handler - The new error event handler.

See Also:

[Parser.setErrorHandler\(org.xml.sax.ErrorHandler\)](#)

parse

```
public void parse(java.lang.String systemId)  
    throws java.io.IOException,  
           SAXException
```

Parse the document.

This method will throw an exception if the embedded XMLReader does not support the <http://xml.org/sax/features/namespace-prefixes> property.

Specified by:

[parse](#) in interface [Parser](#)

Parameters:

`systemId` - The absolute URL of the document.

Throws:

`java.io.IOException` - If there is a problem reading the raw content of the document.

[SAXException](#) - If there is a problem processing the document.

See Also:

[parse\(org.xml.sax.InputSource\)](#), [Parser.parse\(java.lang.String\)](#)

parse

```
public void parse(InputSource input)
    throws java.io.IOException,
           SAXException
```

Parse the document.

This method will throw an exception if the embedded XMLReader does not support the <http://xml.org/sax/features/namespace-prefixes> property.

Specified by:

[parse](#) in interface [Parser](#)

Parameters:

`input` - An input source for the document.

Throws:

`java.io.IOException` - If there is a problem reading the raw content of the document.

[SAXException](#) - If there is a problem processing the document.

See Also:

[parse\(java.lang.String\)](#), [Parser.parse\(org.xml.sax.InputSource\)](#)

setDocumentLocator

```
public void setDocumentLocator(Locator locator)
```

Set a document locator.

Specified by:

[setDocumentLocator](#) in interface [ContentHandler](#)

Parameters:

`locator` - The document locator.

See Also:

[ContentHandler.setDocumentLocator\(org.xml.sax Locator\)](#)

startDocument

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

Start document event.

Specified by:

[startDocument](#) in interface [ContentHandler](#)

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[ContentHandler.startDocument\(\)](#)

endDocument

```
public void endDocument()  
           throws SAXException
```

End document event.

Specified by:

[endDocument](#) in interface [ContentHandler](#)

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[ContentHandler.endDocument\(\)](#)

startPrefixMapping

```
public void startPrefixMapping(java.lang.String prefix,  
                               java.lang.String uri)
```

Adapt a SAX2 start prefix mapping event.

Specified by:

[startPrefixMapping](#) in interface [ContentHandler](#)

Parameters:

`prefix` - The prefix being mapped.

`uri` - The Namespace URI being mapped to.

See Also:

[ContentHandler.startPrefixMapping\(java.lang.String, java.lang.String\)](#)

endPrefixMapping

```
public void endPrefixMapping(java.lang.String prefix)
```

Adapt a SAX2 end prefix mapping event.

Specified by:

[endPrefixMapping](#) in interface [ContentHandler](#)

Parameters:

`prefix` - The prefix being mapped.

See Also:

[ContentHandler.endPrefixMapping\(java.lang.String\)](#)

startElement

```
public void startElement(java.lang.String uri,  
                           java.lang.String localName,  
                           java.lang.String qName,  
                           Attributes atts)  
    throws SAXException
```

Adapt a SAX2 start element event.

Specified by:

[startElement](#) in interface [ContentHandler](#)

Parameters:

`uri` - The Namespace URI.

`localName` - The Namespace local name.

`qName` - The qualified (prefixed) name.

`atts` - The SAX2 attributes.

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[ContentHandler.endDocument\(\)](#)

endElement

```
public void endElement(java.lang.String uri,  
                       java.lang.String localName,  
                       java.lang.String qName)  
    throws SAXException
```

Adapt a SAX2 end element event.

Specified by:

[endElement](#) in interface [ContentHandler](#)

Parameters:

`uri` - The Namespace URI.

`localName` - The Namespace local name.

`qName` - The qualified (prefixed) name.

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[ContentHandler.endElement\(java.lang.String, java.lang.String, java.lang.String\)](#)

characters

```
public void characters(char[] ch,  
                      int start,  
                      int length)  
    throws SAXException
```

Adapt a SAX2 characters event.

Specified by:

[characters](#) in interface [ContentHandler](#)

Parameters:

`ch` - An array of characters.

`start` - The starting position in the array.

`length` - The number of characters to use.

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[ContentHandler.characters\(char\[\], int, int\)](#)

ignorableWhitespace

```
public void ignorableWhitespace(char[] ch,
                                int start,
                                int length)
                                throws SAXException
```

Adapt a SAX2 ignorable whitespace event.

Specified by:

[ignorableWhitespace](#) in interface [ContentHandler](#)

Parameters:

`ch` - An array of characters.

`start` - The starting position in the array.

`length` - The number of characters to use.

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[ContentHandler.ignorableWhitespace\(char\[\], int, int\)](#)

processingInstruction

```
public void processingInstruction(java.lang.String target,
                                java.lang.String data)
                                throws SAXException
```

Adapt a SAX2 processing instruction event.

Specified by:

[processingInstruction](#) in interface [ContentHandler](#)

Parameters:

`target` - The processing instruction target.

data - The remainder of the processing instruction

Throws:

[SAXException](#) - The client may raise a processing exception.

See Also:

[ContentHandler.processingInstruction\(java.lang.String, java.lang.String\)](#)

skippedEntity

```
public void skippedEntity(java.lang.String name)
    throws SAXException
```

Adapt a SAX2 skipped entity event.

Specified by:

[skippedEntity](#) in interface [ContentHandler](#)

Parameters:

name - The name of the skipped entity.

See Also:

[ContentHandler.skippedEntity\(java.lang.String\)](#)

[Overview](#) [Package](#) **Class** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#)SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)`org.xml.sax.helpers`

Class XMLReaderFactory

`java.lang.Object`

```

|
+--org.xml.sax.helpers.XMLReaderFactory

```

public final class XMLReaderFactoryextends `java.lang.Object`

Factory for creating an XML reader.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

This class contains static methods for creating an XML reader from an explicit class name, or for creating an XML reader based on the value of the `org.xml.sax.driver` system property:

```

try {
    XMLReader myReader = XMLReaderFactory.createXMLReader();
} catch (SAXException e) {
    System.err.println(e.getMessage());
}

```

Note that these methods will not be usable in environments where system properties are not accessible or where the application or applet is not permitted to load classes dynamically.

Note to implementors: SAX implementations in specialized environments may replace this class with a different one optimized for the environment, as long as its method signatures remain the same.

Since:

SAX 2.0

Version:

2.0

Author:David Megginson, sax@megginson.com**See Also:**[XMLReader](#)

Method Summary

static XMLReader	createXMLReader () Attempt to create an XML reader from a system property.
static XMLReader	createXMLReader (java.lang.String className) Attempt to create an XML reader from a class name.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

createXMLReader

```
public static XMLReader createXMLReader( )
                                   throws SAXException
```

Attempt to create an XML reader from a system property.

This method uses the value of the system property "org.xml.sax.driver" as the full name of a Java class and tries to instantiate that class as a SAX2 XMLReader.

Note that many Java interpreters allow system properties to be specified on the command line.

Returns:

A new XMLReader.

Throws:

[SAXException](#) - If the value of the "org.xml.sax.driver" system property is null, or if the class cannot be loaded and instantiated.

See Also:

[createXMLReader\(java.lang.String \)](#)

createXMLReader

```
public static XMLReader createXMLReader( java.lang.String className)
                                   throws SAXException
```

Attempt to create an XML reader from a class name.

Given a class name, this method attempts to load and instantiate the class as an XML reader.

Returns:

A new XML reader.

Throws:

[SAXException](#) - If the class cannot be loaded, instantiated, and cast to XMLReader.

See Also:

[createXMLReader\(\)](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)



Document Object Model (DOM) Level 2 Core Specification

Version 1.0

W3C Recommendation 13 November, 2000

This version:

<http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113>
([PostScript file](#) , [PDF file](#) , [plain text](#) , [ZIP file](#))

Latest version:

<http://www.w3.org/TR/DOM-Level-2-Core>

Previous version:

<http://www.w3.org/TR/2000/PR-DOM-Level-2-Core-20000927>

Editors:

Arnaud Le Hors, *W3C team contact until October 1999, then IBM*

Philippe Le Hégaré, *W3C, team contact (from November 1999)*

Lauren Wood, *SoftQuad Software Inc., WG Chair*

Gavin Nicol, *Inso EPS (for DOM Level 1)*

Jonathan Robie, *Texcel Research and Software AG (for DOM Level 1)*

Mike Champion, *ArborText and Software AG (for DOM Level 1 from November 20, 1997)*

Steve Byrne, *JavaSoft (for DOM Level 1 until November 19, 1997)*

Copyright © 2000 [W3C](#)® ([MIT](#), [INRIA](#), [Keio](#)), All Rights Reserved. W3C [liability](#), [trademark](#), [document use](#) and [software licensing](#) rules apply.

Abstract

This specification defines the Document Object Model Level 2 Core, a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content and structure of documents. The Document Object Model Level 2 Core builds on the Document Object Model Level 1 Core.

The DOM Level 2 Core is made of a set of core interfaces to create and manipulate the structure and contents of a document. The Core also contains specialized interfaces dedicated to XML.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a [W3C Recommendation](#). It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document has been produced as part of the [W3C DOM Activity](#). The authors of this document are the DOM Working Group members. Different modules of the Document Object Model have different editors.

Please send general comments about this document to the public mailing list www-dom@w3.org. An [archive](http://lists.w3.org/Archives/Public/www-dom/) is available at <http://lists.w3.org/Archives/Public/www-dom/>.

The English version of this specification is the only normative version. Information about [translations](#) of this document is available at <http://www.w3.org/2000/11/DOM-Level-2-translations>.

The [list of known errors](#) in this document is available at <http://www.w3.org/2000/11/DOM-Level-2-errata>

A list of [current W3C Recommendations and other technical documents](#) can be found at <http://www.w3.org/TR>.

Table of contents

- [Expanded Table of Contents](#)
- [Copyright Notice](#)
- [What is the Document Object Model?](#)
- [1. Document Object Model Core](#)
- [Appendix A: Changes](#)

- [Appendix B: Accessing code point boundaries](#)
- [Appendix C: IDL Definitions](#)
- [Appendix D: Java Language Binding](#)
- [Appendix E: ECMAScript Language Binding](#)
- [Appendix F: Acknowledgements](#)
- [Glossary](#)
- [References](#)
- [Index](#)

[next](#) [contents](#) [index](#)

[W3C]

Document Object Model (DOM) Level 2 Core Specification

Version 1.0

W3C Recommendation 13 November, 2000

This version:

<http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113>
(PostScript file , PDF file , plain text , ZIP file)

Latest version:

<http://www.w3.org/TR/DOM-Level-2-Core>

Previous version:

<http://www.w3.org/TR/2000/PR-DOM-Level-2-Core-20000927>

Editors:

Arnaud Le Hors, W3C team contact until October 1999, then IBM
Philippe Le Hégaré, W3C, team contact (from November 1999)
Lauren Wood, SoftQuad Software Inc., WG Chair
Gavin Nicol, Inso EPS (for DOM Level 1)
Jonathan Robie, Texcel Research and Software AG (for DOM Level 1)
Mike Champion, ArborText and Software AG (for DOM Level 1 from November 20, 1997)
Steve Byrne, JavaSoft (for DOM Level 1 until November 19, 1997)

Copyright © 2000 W3C® (MIT, INRIA, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

Abstract

This specification defines the Document Object Model Level 2 Core, a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content and structure of documents. The Document Object Model Level 2 Core builds on the Document Object Model Level 1 Core.

The DOM Level 2 Core is made of a set of core interfaces to create and manipulate the structure and contents of a document. The Core also contains specialized interfaces dedicated to XML.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document has been produced as part of the W3C DOM Activity. The authors of this document are the DOM Working Group members. Different modules of the Document Object Model have different editors.

Please send general comments about this document to the public mailing list www-dom@w3.org. An archive is available at <http://lists.w3.org/Archives/Public/www-dom/>.

The English version of this specification is the only normative version. Information about translations of this document is available at <http://www.w3.org/2000/11/DOM-Level-2-translations>.

The list of known errors in this document is available at <http://www.w3.org/2000/11/DOM-Level-2-errata>

A list of current W3C Recommendations and other technical documents can be found at <http://www.w3.org/TR>.

Table of contents

- * Expanded Table of Contents
- * Copyright Notice
- * What is the Document Object Model?
-
- * 1. Document Object Model Core
-
- * Appendix A: Changes
- * Appendix B: Accessing code point boundaries
- * Appendix C: IDL Definitions
- * Appendix D: Java Language Binding
- * Appendix E: ECMAScript Language Binding
- * Appendix F: Acknowledgements
- * Glossary
- * References
- * Index

13 November, 2000

Expanded Table of Contents

- * Expanded Table of Contents
- * Copyright Notice
 - o W3C Document Copyright Notice and License
 - o W3C Software Copyright Notice and License
- * What is the Document Object Model?
 - o Introduction
 - o What the Document Object Model is
 - o What the Document Object Model is not
 - o Where the Document Object Model came from
 - o Entities and the DOM Core
 - o Conformance
 - o DOM Interfaces and DOM Implementations
- * 1. Document Object Model Core
 - o 1.1. Overview of the DOM Core Interfaces
 - + 1.1.1. The DOM Structure Model
 - + 1.1.2. Memory Management
 - + 1.1.3. Naming Conventions
 - + 1.1.4. Inheritance vs. Flattened Views of the API
 - + 1.1.5. The DOMString type
 - + 1.1.6. The DOMTimeStamp type
 - + 1.1.7. String comparisons in the DOM
 - + 1.1.8. XML Namespaces
 - o 1.2. Fundamental Interfaces
 - o 1.3. Extended Interfaces

- * Appendix A: Changes
 - o A.1. Changes between DOM Level 1 Core and DOM Level 2 Core
 - + A.1.1. Changes to DOM Level 1 Core interfaces and exceptions
 - + A.1.2. New features
- * Appendix B: Accessing code point boundaries
 - o B.1. Introduction
 - o B.2. Methods
- * Appendix C: IDL Definitions
- * Appendix D: Java Language Binding
- * Appendix E: ECMAScript Language Binding
- * Appendix F: Acknowledgements
 - o F.1. Production Systems
- * Glossary
- * References
 - o 1. Normative references
 - o 2. Informative references
- * Index

13 November, 2000

Copyright Notice

Copyright © 2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

This document is published under the W3C Document Copyright Notice and License. The bindings within this document are published under the W3C Software Copyright Notice and License. The software license requires "Notice of any changes or modifications to the W3C files, including the date changes were made." Consequently, modified versions of the DOM bindings must document that they do not conform to the W3C standard; in the case of the IDL definitions, the pragma prefix can no longer be 'w3c.org'; in the case of the Java Language binding, the package names can no longer be in the 'org.w3c' package.

W3C Document Copyright Notice and License

Note: This section is a copy of the W3C Document Notice and License and could be found at
<http://www.w3.org/Consortium/Legal/copyright-documents-19990405>.

Copyright © 1994-2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

<http://www.w3.org/Consortium/Legal/>

Public documents on the W3C site are provided by the copyright holders under the following license. The software or Document Type Definitions (DTDs) associated with W3C specifications are governed by the Software Notice. By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you

include the following on ALL copies of the document, or portions thereof, that you use:

1. A link or URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it doesn't exist, a notice of the form: "Copyright © [\$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. <http://www.w3.org/Consortium/Legal/>" (Hypertext is preferred, but a textual representation is permitted.)
3. If it exists, the STATUS of the W3C document.

When space permits, inclusion of the full text of this NOTICE should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the Copyright FAQ) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

W3C Software Copyright Notice and License

Note: This section is a copy of the W3C Software Copyright Notice and License and could be found at
<http://www.w3.org/Consortium/Legal/copyright-software-19980720>

Copyright © 1994-2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

<http://www.w3.org/Consortium/Legal/>

This W3C work (including software, documents, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and modify this software and its documentation, with or without modification, for any purpose and without fee or royalty is

hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications, that you make:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers. If none exist, then a notice of the following form: "Copyright © [\$date-of-software] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. <http://www.w3.org/Consortium/Legal/>."
3. Notice of any changes or modifications to the W3C files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

13 November, 2000

What is the Document Object Model?

Editors

Philippe Le Hégaré, W3C
Lauren Wood, SoftQuad Software Inc., WG Chair
Jonathan Robie, Texcel (for DOM Level 1)

Introduction

The Document Object Model (DOM) is an application programming interface (API) for valid HTML and well-formed XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions - in particular, the DOM interfaces for the XML internal and external subsets have not yet been specified.

As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The DOM is designed to be

used with any programming language. In order to provide a precise, language-independent specification of the DOM interfaces, we have chosen to define the specifications in Object Management Group (OMG) IDL [OMGIDL], as defined in the CORBA 2.3.1 specification [CORBA]. In addition to the OMG IDL specification, we provide language bindings for Java [Java] and ECMAScript [ECMAScript] (an industry-standard scripting language based on JavaScript [JavaScript] and JScript [JScript]).

Note: OMG IDL is used only as a language-independent and implementation-neutral way to specify interfaces. Various other IDLs could have been used ([COM], [JavaIDL], [MIDL], ...). In general, IDLs are designed for specific computing environments. The Document Object Model can be implemented in any computing environment, and does not require the object binding runtimes generally associated with such IDLs.

What the Document Object Model is

The DOM is a programming API for documents. It is based on an object structure that closely resembles the structure of the documents it models. For instance, consider this table, taken from an HTML document:

```
<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</TBODY>
</TABLE>
```

A graphical representation of the DOM of the example table is:

```
-----
[graphical representation of the DOM of the example table]
-----
graphical representation of the DOM of the example table
-----
```

In the DOM, documents have a logical structure which is very much like a tree; to be more precise, which is like a "forest" or "grove", which can contain more than one tree. Each document contains zero or one doctype nodes, one root element node, and zero or more comments or processing instructions; the root element serves as the root of the element tree for the document. However, the DOM does not specify that documents must be implemented as a tree or a grove, nor does it specify how the relationships among objects be implemented. The DOM is a logical model that may be implemented in any convenient manner. In this specification, we use the term structure model to describe the tree-like representation of a document. We also use the term "tree" when referring to the arrangement of those information items which can be reached by using "tree-walking" methods; (this does not include attributes). One important property of DOM structure models is structural isomorphism: if any two Document Object Model implementations are used to create a representation of the same document, they will create the same structure model, in accordance with the XML Information Set [InfoSet].

Note: There may be some variations depending on the parser being used to build the DOM. For instance, the DOM may not contain whitespaces in element content if the parser discards them.

The name "Document Object Model" was chosen because it is an "object model" in the traditional object oriented design sense: documents are modeled using objects, and the model encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed. In other words, the nodes in the above diagram do not represent a data structure, they represent objects, which have functions and identity. As an object model, the DOM identifies:

- * the interfaces and objects used to represent and manipulate a document
- * the semantics of these interfaces and objects - including both behavior and attributes
- * the relationships and collaborations among these interfaces and objects

The structure of SGML documents has traditionally been represented by an abstract data model, not by an object model. In an abstract data model, the model is centered around the data. In object oriented programming languages, the data itself is encapsulated in objects that hide the data, protecting it from direct external manipulation. The functions associated with these objects determine how the objects may be manipulated, and they are part of the object model.

What the Document Object Model is not

This section is designed to give a more precise understanding of the DOM by distinguishing it from other systems that may seem to be like it.

- * The Document Object Model is not a binary specification. DOM programs written in the same language binding will be source code compatible across platforms, but the DOM does not define any form of binary interoperability.
- * The Document Object Model is not a way of persisting objects to XML or HTML. Instead of specifying how objects may be represented in XML, the DOM specifies how XML and HTML documents are represented as objects, so that they may be used in object oriented programs.
- * The Document Object Model is not a set of data structures; it is an object model that specifies interfaces. Although this document contains diagrams showing parent/child relationships, these are logical relationships defined by the programming interfaces, not representations of any particular internal data structures.
- * The Document Object Model does not define what information in a document is relevant or how information in a document is structured. For XML, this is specified by the W3C XML Information Set [Infoset]. The DOM is simply an API to this information set.
- * The Document Object Model, despite its name, is not a competitor to the Component Object Model (COM). COM, like CORBA, is a language independent way to specify interfaces and objects; the DOM is a set of interfaces and objects designed for managing HTML and XML documents. The DOM may be implemented using language-independent systems like COM or CORBA; it may also be implemented using language-specific bindings like the Java or ECMAScript bindings specified in this document.

Where the Document Object Model came from

The DOM originated as a specification to allow JavaScript scripts and Java programs to be portable among Web browsers. "Dynamic HTML" was the immediate ancestor of the Document Object Model, and it was originally thought of largely in terms of browsers. However, when the DOM Working Group was formed

at W3C, it was also joined by vendors in other domains, including HTML or XML editors and document repositories. Several of these vendors had worked with SGML before XML was developed; as a result, the DOM has been influenced by SGML Groves and the HyTime standard. Some of these vendors had also developed their own object models for documents in order to provide an API for SGML/XML editors or document repositories, and these object models have also influenced the DOM.

Entities and the DOM Core

In the fundamental DOM interfaces, there are no objects representing entities. Numeric character references, and references to the pre-defined entities in HTML and XML, are replaced by the single character that makes up the entity's replacement. For example, in:

```
<p>This is a dog & a cat</p>
```

the "&" will be replaced by the character "&", and the text in the P element will form a single continuous sequence of characters. Since numeric character references and pre-defined entities are not recognized as such in CDATA sections, or in the SCRIPT and STYLE elements in HTML, they are not replaced by the single character they appear to refer to. If the example above were enclosed in a CDATA section, the "&" would not be replaced by "&"; neither would the <p> be recognized as a start tag. The representation of general entities, both internal and external, are defined within the extended (XML) interfaces of DOM Level 1 [DOM Level 1].

Note: When a DOM representation of a document is serialized as XML or HTML text, applications will need to check each character in text data to see if it needs to be escaped using a numeric or pre-defined entity. Failing to do so could result in invalid HTML or XML. Also, implementations should be aware of the fact that serialization into a character encoding ("charset") that does not fully cover ISO 10646 may fail if there are characters in markup or CDATA sections that are not present in the encoding.

Conformance

This section explains the different levels of conformance to DOM Level 2. DOM Level 2 consists of 14 modules. It is possible to conform to DOM Level 2, or to a DOM Level 2 module.

An implementation is DOM Level 2 conformant if it supports the Core module defined in this document (see Fundamental Interfaces). An implementation conforms to a DOM Level 2 module if it supports all the interfaces for that module and the associated semantics.

Here is the complete list of DOM Level 2.0 modules and the features used by them. Feature names are case-insensitive.

Core module

defines the feature "Core".

XML module

defines the feature "XML".

HTML module

defines the feature "HTML". (see [DOM Level 2 HTML]).

Note: At time of publication, this DOM Level 2 module is not yet a W3C Recommendation.

Views module

defines the feature "Views" in [DOM Level 2 Views].

Style Sheets module

defines the feature "StyleSheets" in [DOM Level 2 Style Sheets].

CSS module

defines the feature "CSS" in [DOM Level 2 CSS].

CSS2 module

defines the feature "CSS2" in [DOM Level 2 CSS].

Events module

defines the feature "Events" in [DOM Level 2 Events].

User interface Events module

defines the feature "UIEvents" in [DOM Level 2 Events].

Mouse Events module

defines the feature "MouseEvents" in [DOM Level 2 Events].

Mutation Events module

defines the feature "MutationEvents" in [DOM Level 2 Events].

HTML Events module

defines the feature "HTMLEvents" in [DOM Level 2 Events].

Range module

defines the feature "Range" in [DOM Level 2 Range].

Traversal module

defines the feature "Traversal" in [DOM Level 2 Traversal].

A DOM implementation must not return "true" to the `hasFeature(feature, version)` method of the `DOMImplementation` interface for that feature unless the implementation conforms to that module. The version number for all features used in DOM Level 2.0 is "2.0".

DOM Interfaces and DOM Implementations

The DOM specifies interfaces which may be used to manage XML or HTML documents. It is important to realize that these interfaces are an abstraction - much like "abstract base classes" in C++, they are a means of specifying a way to access and manipulate an application's internal representation of a document. Interfaces do not imply a particular concrete implementation. Each DOM application is free to maintain documents in any convenient representation, as long as the interfaces shown in this specification are supported. Some DOM implementations will be existing programs that use the DOM interfaces to access software written long before the DOM specification existed. Therefore, the DOM is designed to avoid implementation dependencies; in particular,

1. Attributes defined in the IDL do not imply concrete objects which must have specific data members - in the language bindings, they are translated to a pair of `get()/set()` functions, not to a data member. Read-only attributes have only a `get()` function in the language bindings.
2. DOM applications may provide additional interfaces and objects not found in this specification and still be considered DOM conformant.
3. Because we specify interfaces and not the actual objects that are to be created, the DOM cannot know what constructors to call for an implementation. In general, DOM users call the `createX()` methods on the `Document` class to create document structures, and DOM implementations create their own internal representations of these structures in their implementations of the `createX()` functions.

The Level 1 interfaces were extended to provide both Level 1 and Level 2 functionality.

DOM implementations in languages other than Java or ECMAScript may choose bindings that are appropriate and natural for their language and run time environment. For example, some systems may need to create a `Document2` class

which inherits from Document and contains the new methods and attributes.

DOM Level 2 does not specify multithreading mechanisms.

13 November, 2000

1. Document Object Model Core

Editors

Arnaud Le Hors, IBM

Gavin Nicol, Inso EPS (for DOM Level 1)

Lauren Wood, SoftQuad, Inc. (for DOM Level 1)

Mike Champion, ArborText (for DOM Level 1 from November 20, 1997)

Steve Byrne, JavaSoft (for DOM Level 1 until November 19, 1997)

Table of contents

- * 1.1. Overview of the DOM Core Interfaces
 - o 1.1.1. The DOM Structure Model
 - o 1.1.2. Memory Management
 - o 1.1.3. Naming Conventions
 - o 1.1.4. Inheritance vs. Flattened Views of the API
 - o 1.1.5. The DOMString type
 - o 1.1.6. The DOMTimeStamp type
 - o 1.1.7. String comparisons in the DOM
 - o 1.1.8. XML Namespaces
- * 1.2. Fundamental Interfaces
 - o DOMException, ExceptionCode, DOMImplementation, DocumentFragment, Document, Node, NodeList, NamedNodeMap, CharacterData, Attr, Element, Text, Comment
- * 1.3. Extended Interfaces
 - o CDATASection, DocumentType, Notation, Entity, EntityReference, ProcessingInstruction

1.1. Overview of the DOM Core Interfaces

This section defines a set of objects and interfaces for accessing and manipulating document objects. The functionality specified in this section (the Core functionality) is sufficient to allow software developers and web script authors to access and manipulate parsed HTML and XML content inside conforming products. The DOM Core API also allows creation and population of a Document object using only DOM API calls; loading a Document and saving it persistently is left to the product that implements the DOM API.

1.1.1. The DOM Structure Model

The DOM presents documents as a hierarchy of Node objects that also implement other, more specialized interfaces. Some types of nodes may have child nodes of various types, and others are leaf nodes that cannot have anything below them in the document structure. For XML and HTML, the node types, and which node types they may have as children, are as follows:

- * Document -- Element (maximum of one), ProcessingInstruction, Comment, DocumentType (maximum of one)
- * DocumentFragment -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- * DocumentType -- no children
- * EntityReference -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- * Element -- Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference

- * Attr -- Text, EntityReference
- * ProcessingInstruction -- no children
- * Comment -- no children
- * Text -- no children
- * CDATASection -- no children
- * Entity -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- * Notation -- no children

The DOM also specifies a `NodeList` interface to handle ordered lists of Nodes, such as the children of a Node, or the elements returned by the `getElementsByTagName` method of the `Element` interface, and also a `NamedNodeMap` interface to handle unordered sets of nodes referenced by their name attribute, such as the attributes of an `Element`. `NodeList` and `NamedNodeMap` objects in the DOM are live; that is, changes to the underlying document structure are reflected in all relevant `NodeList` and `NamedNodeMap` objects. For example, if a DOM user gets a `NodeList` object containing the children of an `Element`, then subsequently adds more children to that element (or removes children, or modifies them), those changes are automatically reflected in the `NodeList`, without further action on the user's part. Likewise, changes to a Node in the tree are reflected in all references to that Node in `NodeList` and `NamedNodeMap` objects.

Finally, the interfaces `Text`, `Comment`, and `CDATASection` all inherit from the `CharacterData` interface.

1.1.2. Memory Management

Most of the APIs defined by this specification are interfaces rather than classes. That means that an implementation need only expose methods with the defined names and specified operation, not implement classes that correspond directly to the interfaces. This allows the DOM APIs to be implemented as a thin veneer on top of legacy applications with their own data structures, or on top of newer applications with different class hierarchies. This also means that ordinary constructors (in the Java or C++ sense) cannot be used to create DOM objects, since the underlying objects to be constructed may have little relationship to the DOM interfaces. The conventional solution to this in object-oriented design is to define factory methods that create instances of objects that implement the various interfaces. Objects implementing some interface "X" are created by a "createX()" method on the `Document` interface; this is because all DOM objects live in the context of a specific `Document`.

The DOM Level 2 API does not define a standard way to create `DOMImplementation` objects; DOM implementations must provide some proprietary way of bootstrapping these DOM interfaces, and then all other objects can be built from there.

The Core DOM APIs are designed to be compatible with a wide range of languages, including both general-user scripting languages and the more challenging languages used mostly by professional programmers. Thus, the DOM APIs need to operate across a variety of memory management philosophies, from language bindings that do not expose memory management to the user at all, through those (notably Java) that provide explicit constructors but provide an automatic garbage collection mechanism to automatically reclaim unused memory, to those (especially C/C++) that generally require the programmer to explicitly allocate object memory, track where it is used, and explicitly free it for re-use. To ensure a consistent API across these platforms, the DOM does not address memory management issues at all, but instead leaves these for the implementation. Neither of the explicit language bindings defined by the DOM API (for ECMAScript and Java) require

any memory management methods, but DOM bindings for other languages (especially C or C++) may require such support. These extensions will be the responsibility of those adapting the DOM API to a specific language, not the DOM Working Group.

1.1.3. Naming Conventions

While it would be nice to have attribute and method names that are short, informative, internally consistent, and familiar to users of similar APIs, the names also should not clash with the names in legacy APIs supported by DOM implementations. Furthermore, both OMG IDL and ECMAScript have significant limitations in their ability to disambiguate names from different namespaces that make it difficult to avoid naming conflicts with short, familiar names. So, DOM names tend to be long and descriptive in order to be unique across all environments.

The Working Group has also attempted to be internally consistent in its use of various terms, even though these may not be common distinctions in other APIs. For example, the DOM API uses the method name "remove" when the method changes the structural model, and the method name "delete" when the method gets rid of something inside the structure model. The thing that is deleted is not returned. The thing that is removed may be returned, when it makes sense to return it.

1.1.4. Inheritance vs. Flattened Views of the API

The DOM Core APIs present two somewhat different sets of interfaces to an XML/HTML document: one presenting an "object oriented" approach with a hierarchy of inheritance, and a "simplified" view that allows all manipulation to be done via the Node interface without requiring casts (in Java and other C-like languages) or query interface calls in COM environments. These operations are fairly expensive in Java and COM, and the DOM may be used in performance-critical environments, so we allow significant functionality using just the Node interface. Because many other users will find the inheritance hierarchy easier to understand than the "everything is a Node" approach to the DOM, we also support the full higher-level interfaces for those who prefer a more object-oriented API.

In practice, this means that there is a certain amount of redundancy in the API. The Working Group considers the "inheritance" approach the primary view of the API, and the full set of functionality on Node to be "extra" functionality that users may employ, but that does not eliminate the need for methods on other interfaces that an object-oriented analysis would dictate. (Of course, when the O-O analysis yields an attribute or method that is identical to one on the Node interface, we don't specify a completely redundant one.) Thus, even though there is a generic nodeName attribute on the Node interface, there is still a tagName attribute on the Element interface; these two attributes must contain the same value, but the it is worthwhile to support both, given the different constituencies the DOM API must satisfy.

1.1.5. The DOMString type

To ensure interoperability, the DOM specifies the following:

* Type Definition DOMString

A DOMString is a sequence of 16-bit units.

IDL Definition

```
valuetype DOMString sequence<unsigned short>;
```

- * Applications must encode DOMString using UTF-16 (defined in [Unicode] and Amendment 1 of [ISO/IEC 10646]).
The UTF-16 encoding was chosen because of its widespread industry practice. Note that for both HTML and XML, the document character set (and therefore the notation of numeric character references) is based on UCS [ISO-10646]. A single numeric character reference in a source document may therefore in some cases correspond to two 16-bit units in a DOMString (a high surrogate and a low surrogate).

Note: Even though the DOM defines the name of the string type to be DOMString, bindings may use different names. For example for Java, DOMString is bound to the String type because it also uses UTF-16 as its encoding.

Note: As of August 2000, the OMG IDL specification ([OMGIDL]) included a wstring type. However, that definition did not meet the interoperability criteria of the DOM API since it relied on negotiation to decide the width and encoding of a character.

1.1.6. The DOMTimeStamp type

To ensure interoperability, the DOM specifies the following:

- * Type Definition DOMTimeStamp

A DOMTimeStamp represents a number of milliseconds.

IDL Definition

```
typedef unsigned long long DOMTimeStamp;
```

- * Note: Even though the DOM uses the type DOMTimeStamp, bindings may use different types. For example for Java, DOMTimeStamp is bound to the long type. In ECMAScript, TimeStamp is bound to the Date type because the range of the integer type is too small.

1.1.7. String comparisons in the DOM

The DOM has many interfaces that imply string matching. HTML processors generally assume an uppercase (less often, lowercase) normalization of names for such things as elements, while XML is explicitly case sensitive. For the purposes of the DOM, string matching is performed purely by binary comparison of the 16-bit units of the DOMString. In addition, the DOM assumes that any case normalizations take place in the processor, before the DOM structures are built.

Note: Besides case folding, there are additional normalizations that can be applied to text. The W3C I18N Working Group is in the process of defining exactly which normalizations are necessary, and where they should be applied. The W3C I18N Working Group expects to require early normalization, which means that data read into the DOM is assumed to already be normalized. The DOM and applications built on top of it in this case only have to assure that text remains normalized when being changed. For further details, please see [Charmod].

1.1.8. XML Namespaces

The DOM Level 2 supports XML namespaces [Namespaces] by augmenting several interfaces of the DOM Level 1 Core to allow creating and manipulating

elements and attributes associated to a namespace.

As far as the DOM is concerned, special attributes used for declaring XML namespaces are still exposed and can be manipulated just like any other attribute. However, nodes are permanently bound to namespace URIs as they get created. Consequently, moving a node within a document, using the DOM, in no case results in a change of its namespace prefix or namespace URI. Similarly, creating a node with a namespace prefix and namespace URI, or changing the namespace prefix of a node, does not result in any addition, removal, or modification of any special attributes for declaring the appropriate XML namespaces. Namespace validation is not enforced; the DOM application is responsible. In particular, since the mapping between prefixes and namespace URIs is not enforced, in general, the resulting document cannot be serialized naively. For example, applications may have to declare every namespace in use when serializing a document.

DOM Level 2 doesn't perform any URI normalization or canonicalization. The URIs given to the DOM are assumed to be valid (e.g., characters such as whitespaces are properly escaped), and no lexical checking is performed. Absolute URI references are treated as strings and compared literally. How relative namespace URI references are treated is undefined. To ensure interoperability only absolute namespace URI references (i.e., URI references beginning with a scheme name and a colon) should be used. Note that because the DOM does no lexical checking, the empty string will be treated as a real namespace URI in DOM Level 2 methods. Applications must use the value null as the namespaceURI parameter for methods if they wish to have no namespace.

Note: In the DOM, all namespace declaration attributes are by definition bound to the namespace URI: "http://www.w3.org/2000/xmlns/". These are the attributes whose namespace prefix or qualified name is "xmlns". Although, at the time of writing, this is not part of the XML Namespaces specification [Namespaces], it is planned to be incorporated in a future revision.

In a document with no namespaces, the child list of an EntityReference node is always the same as that of the corresponding Entity. This is not true in a document where an entity contains unbound namespace prefixes. In such a case, the descendants of the corresponding EntityReference nodes may be bound to different namespace URIs, depending on where the entity references are. Also, because, in the DOM, nodes always remain bound to the same namespace URI, moving such EntityReference nodes can lead to documents that cannot be serialized. This is also true when the DOM Level 1 method createEntityReference of the Document interface is used to create entity references that correspond to such entities, since the descendants of the returned EntityReference are unbound. The DOM Level 2 does not support any mechanism to resolve namespace prefixes. For all of these reasons, use of such entities and entity references should be avoided or used with extreme care. A future Level of the DOM may include some additional support for handling these.

The new methods, such as createElementNS and createAttributeNS of the Document interface, are meant to be used by namespace aware applications. Simple applications that do not use namespaces can use the DOM Level 1 methods, such as createElement and createAttribute. Elements and attributes created in this way do not have any namespace prefix, namespace URI, or local name.

Note: DOM Level 1 methods are namespace ignorant. Therefore, while it is safe to use these methods when not dealing with namespaces, using them and the new ones at the same time should be avoided. DOM Level 1 methods solely identify attribute nodes by their nodeName. On the contrary, the DOM Level 2

methods related to namespaces, identify attribute nodes by their namespaceURI and localName. Because of this fundamental difference, mixing both sets of methods can lead to unpredictable results. In particular, using setAttributeNS, an element may have two attributes (or more) that have the same nodeName, but different namespaceURIs. Calling getAttribute with that nodeName could then return any of those attributes. The result depends on the implementation. Similarly, using setAttributeNode, one can set two attributes (or more) that have different nodeNames but the same prefix and namespaceURI. In this case getAttributeNodeNS will return either attribute, in an implementation dependent manner. The only guarantee in such cases is that all methods that access a named item by its nodeName will access the same item, and all methods which access a node by its URI and local name will access the same node. For instance, setAttribute and setAttributeNS affect the node that getAttribute and getAttributeNS, respectively, return.

1.2. Fundamental Interfaces

The interfaces within this section are considered fundamental, and must be fully implemented by all conforming implementations of the DOM, including all HTML DOM implementations [DOM Level 2 HTML], unless otherwise specified.

A DOM application may use the hasFeature(feature, version) method of the DOMImplementation interface with parameter values "Core" and "2.0" (respectively) to determine whether or not this module is supported by the implementation. Any implementation that conforms to DOM Level 2 or a DOM Level 2 module must conform to the Core module. Please refer to additional information about conformance in this specification.

Exception DOMException

DOM operations only raise exceptions in "exceptional" circumstances, i.e., when an operation is impossible to perform (either for logical reasons, because data is lost, or because the implementation has become unstable). In general, DOM methods return specific error values in ordinary processing situations, such as out-of-bound errors when using NodeList.

Implementations should raise other exceptions under other circumstances. For example, implementations should raise an implementation-dependent exception if a null argument is passed.

Some languages and object systems do not support the concept of exceptions. For such systems, error conditions may be indicated using native error reporting mechanisms. For some bindings, for example, methods may return error codes similar to those listed in the corresponding method descriptions.

IDL Definition

```
exception DOMException {
    unsigned short    code;
};
// ExceptionCode
const unsigned short    INDEX_SIZE_ERR           = 1;
const unsigned short    DOMSTRING_SIZE_ERR       = 2;
const unsigned short    HIERARCHY_REQUEST_ERR    = 3;
const unsigned short    WRONG_DOCUMENT_ERR       = 4;
const unsigned short    INVALID_CHARACTER_ERR    = 5;
const unsigned short    NO_DATA_ALLOWED_ERR      = 6;
const unsigned short    NO_MODIFICATION_ALLOWED_ERR = 7;
const unsigned short    NOT_FOUND_ERR            = 8;
```



```
const unsigned short    NOT_SUPPORTED_ERR          = 9;
const unsigned short    INUSE_ATTRIBUTE_ERR        = 10;
// Introduced in DOM Level 2:
const unsigned short    INVALID_STATE_ERR          = 11;
// Introduced in DOM Level 2:
const unsigned short    SYNTAX_ERR                 = 12;
// Introduced in DOM Level 2:
const unsigned short    INVALID_MODIFICATION_ERR   = 13;
// Introduced in DOM Level 2:
const unsigned short    NAMESPACE_ERR             = 14;
// Introduced in DOM Level 2:
const unsigned short    INVALID_ACCESS_ERR         = 15;
```

Definition group ExceptionCode

An integer indicating the type of error generated.

Note: Other numeric codes are reserved for W3C for possible future use.

Defined Constants

DOMSTRING_SIZE_ERR
If the specified range of text does not fit into a DOMString

HIERARCHY_REQUEST_ERR
If any node is inserted somewhere it doesn't belong

INDEX_SIZE_ERR
If index or size is negative, or greater than the allowed value

INUSE_ATTRIBUTE_ERR
If an attempt is made to add an attribute that is already in use elsewhere

INVALID_ACCESS_ERR, introduced in DOM Level 2.
If a parameter or an operation is not supported by the underlying object.

INVALID_CHARACTER_ERR
If an invalid or illegal character is specified, such as in a name. See production 2 in the XML specification for the definition of a legal character, and production 5 for the definition of a legal name character.

INVALID_MODIFICATION_ERR, introduced in DOM Level 2.
If an attempt is made to modify the type of the underlying object.

INVALID_STATE_ERR, introduced in DOM Level 2.
If an attempt is made to use an object that is not, or is no longer, usable.

NAMESPACE_ERR, introduced in DOM Level 2.
If an attempt is made to create or change an object in a way which is incorrect with regard to namespaces.

NOT_FOUND_ERR
If an attempt is made to reference a node in a context where it does not exist

NOT_SUPPORTED_ERR
If the implementation does not support the requested type of object or operation.

NO_DATA_ALLOWED_ERR
If data is specified for a node which does not support data

NO_MODIFICATION_ALLOWED_ERR
If an attempt is made to modify an object where modifications are not allowed

SYNTAX_ERR, introduced in DOM Level 2.

If an invalid or illegal string is specified.

WRONG_DOCUMENT_ERR

If a node is used in a different document than the one that created it (that doesn't support it)

Interface DOMImplementation

The DOMImplementation interface provides a number of methods for performing operations that are independent of any particular instance of the document object model.

IDL Definition

```
interface DOMImplementation {
    boolean                hasFeature(in DOMString feature,
                                      in DOMString version);

    // Introduced in DOM Level 2:
    DocumentType            createDocumentType(in DOMString qualifiedName,
                                              in DOMString publicId,
                                              in DOMString systemId)
                                              raises(DOMException);

    // Introduced in DOM Level 2:
    Document                createDocument(in DOMString namespaceURI,
                                          in DOMString qualifiedName,
                                          in DocumentType doctype)
                                          raises(DOMException);
};
```

Methods

createDocument introduced in DOM Level 2

Creates an XML Document object of the specified type with its document element. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString

The namespace URI of the document element to create.

qualifiedName of type DOMString

The qualified name of the document element to be created.

doctype of type DocumentType

The type of document to be created or null.

When doctype is not null, its Node.ownerDocument attribute is set to the document being created.

Return Value

Document A new Document object.

Exceptions

DOMException INVALID_CHARACTER_ERR: Raised if the specified qualified name contains an illegal character.

NAMESPACE_ERR: Raised if the qualifiedName is malformed, if the qualifiedName has a prefix and the namespaceURI is null, or if the qualifiedName has a prefix that is "xml" and the namespaceURI is different from "http://www.w3.org/XML/1998/namespace" [Namespaces].

WRONG_DOCUMENT_ERR: Raised if doctype has already been used with a different document or was created from a different implementation.

createDocumentType introduced in DOM Level 2

Creates an empty `DocumentType` node. Entity declarations and notations are not made available. Entity reference expansions and default attribute additions do not occur. It is expected that a future version of the DOM will provide a way for populating a `DocumentType`.

HTML-only DOM implementations do not need to implement this method.

Parameters

`qualifiedName` of type `DOMString`

The qualified name of the document type to be created.

`publicId` of type `DOMString`

The external subset public identifier.

`systemId` of type `DOMString`

The external subset system identifier.

Return Value

`DocumentType` A new `DocumentType` node with `Node.ownerDocument` set to null.

Exceptions

`DOMException` `INVALID_CHARACTER_ERR`: Raised if the specified qualified name contains an illegal character.

`NAMESPACE_ERR`: Raised if the `qualifiedName` is malformed.

`hasFeature`

Test if the DOM implementation implements a specific feature.

Parameters

`feature` of type `DOMString`

The name of the feature to test (case-insensitive). The values used by DOM features are defined throughout the DOM Level 2 specifications and listed in the Conformance section. The name must be an XML name. To avoid possible conflicts, as a convention, names referring to features defined outside the DOM specification should be made unique by reversing the name of the Internet domain name of the person (or the organization that the person belongs to) who defines the feature, component by component, and using this as a prefix. For instance, the W3C SVG Working Group defines the feature `"org.w3c.dom.svg"`.

`version` of type `DOMString`

This is the version number of the feature to test. In Level 2, the string can be either `"2.0"` or `"1.0"`. If the version is not specified, supporting any version of the feature causes the method to return true.

Return Value

boolean true if the feature is implemented in the specified version, false otherwise.

No Exceptions

Interface `DocumentFragment`

`DocumentFragment` is a "lightweight" or "minimal" `Document` object. It is very common to want to be able to extract a portion of a document's tree or to create a new fragment of a document. Imagine implementing a user command like cut or rearranging a document by moving fragments around. It is desirable to have an object which can hold such fragments and it is quite natural to use a `Node` for this purpose. While it is true that a `Document` object could fulfill this role, a `Document` object can potentially be a heavyweight object, depending on the underlying implementation. What is really needed for this is a very lightweight object. `DocumentFragment` is such an object.

Furthermore, various operations -- such as inserting nodes as children of another Node -- may take DocumentFragment objects as arguments; this results in all the child nodes of the DocumentFragment being moved to the child list of this node.

The children of a DocumentFragment node are zero or more nodes representing the tops of any sub-trees defining the structure of the document. DocumentFragment nodes do not need to be well-formed XML documents (although they do need to follow the rules imposed upon well-formed XML parsed entities, which can have multiple top nodes). For example, a DocumentFragment might have only one child and that child node could be a Text node. Such a structure model represents neither an HTML document nor a well-formed XML document.

When a DocumentFragment is inserted into a Document (or indeed any other Node that may take children) the children of the DocumentFragment and not the DocumentFragment itself are inserted into the Node. This makes the DocumentFragment very useful when the user wishes to create nodes that are siblings; the DocumentFragment acts as the parent of these nodes so that the user can use the standard methods from the Node interface, such as insertBefore and appendChild.

IDL Definition

```
interface DocumentFragment : Node {  
};
```

Interface Document

The Document interface represents the entire HTML or XML document. Conceptually, it is the root of the document tree, and provides the primary access to the document's data.

Since elements, text nodes, comments, processing instructions, etc. cannot exist outside the context of a Document, the Document interface also contains the factory methods needed to create these objects. The Node objects created have a ownerDocument attribute which associates them with the Document within whose context they were created.

IDL Definition

```
interface Document : Node {  
    readonly attribute DocumentType      doctype;  
    readonly attribute DOMImplementation implementation;  
    readonly attribute Element            documentElement;  
    Element                               createElement(in DOMString tagName)  
                                           raises(DOMException);  
    DocumentFragment                     createDocumentFragment();  
    Text                                 createTextNode(in DOMString data);  
    Comment                              createComment(in DOMString data);  
    CDATASection                         createCDATASection(in DOMString data)  
                                           raises(DOMException);  
    ProcessingInstruction                 createProcessingInstruction(in DOMString target,  
                                                                    in DOMString data)  
                                           raises(DOMException);  
    Attr                                 createAttribute(in DOMString name)  
                                           raises(DOMException);  
    EntityReference                      createEntityReference(in DOMString name)  
                                           raises(DOMException);  
    NodeList                             getElementsByTagName(in DOMString tagname);  
    // Introduced in DOM Level 2:
```

```
Node                importNode(in Node importedNode,
                                in boolean deep)
                                raises(DOMException);

// Introduced in DOM Level 2:
Element             createElementNS(in DOMString namespaceURI,
                                    in DOMString qualifiedName)
                                    raises(DOMException);

// Introduced in DOM Level 2:
Attr                createAttributeNS(in DOMString namespaceURI,
                                      in DOMString qualifiedName)
                                      raises(DOMException);

// Introduced in DOM Level 2:
NodeList             getElementByTagNameNS(in DOMString namespaceURI,
                                            in DOMString localName);

// Introduced in DOM Level 2:
Element             getElementById(in DOMString elementId);
};
```

Attributes

doctype of type `DocumentType`, readonly
The Document Type Declaration (see `DocumentType`) associated with this document. For HTML documents as well as XML documents without a document type declaration this returns null. The DOM Level 2 does not support editing the Document Type Declaration. `docType` cannot be altered in any way, including through the use of methods inherited from the `Node` interface, such as `insertNode` or `removeNode`.

documentElement of type `Element`, readonly
This is a convenience attribute that allows direct access to the child node that is the root element of the document. For HTML documents, this is the element with the tagName "HTML".

implementation of type `DOMImplementation`, readonly
The `DOMImplementation` object that handles this document. A DOM application may use objects from multiple implementations.

Methods

createAttribute
Creates an `Attr` of the given name. Note that the `Attr` instance can then be set on an `Element` using the `setAttributeNode` method.
To create an attribute with a qualified name and namespace URI, use the `createAttributeNS` method.
Parameters
`name` of type `DOMString`
The name of the attribute.
Return Value
`Attr` A new `Attr` object with the `nodeName` attribute set to `name`, and `localName`, `prefix`, and `namespaceURI` set to null. The value of the attribute is the empty string.
Exceptions
`DOMException` `INVALID_CHARACTER_ERR`: Raised if the specified name contains an illegal character.

createAttributeNS introduced in DOM Level 2
Creates an attribute of the given qualified name and namespace URI. HTML-only DOM implementations do not need to implement this method.
Parameters
`namespaceURI` of type `DOMString`
The namespace URI of the attribute to create.
`qualifiedName` of type `DOMString`
The qualified name of the attribute to instantiate.

Return Value

Attr A new Attr object with the following attributes:

Attribute	Value
Node.nodeName	qualifiedName
Node.namespaceURI	namespaceURI
Node.prefix	prefix, extracted from qualifiedName, or null if there is no prefix
Node.localName	local name, extracted from qualifiedName
Attr.name	qualifiedName
Node.nodeValue	the empty string

Exceptions

DOMException INVALID_CHARACTER_ERR: Raised if the specified qualified name contains an illegal character.

NAMESPACE_ERR: Raised if the qualifiedName is malformed, if the qualifiedName has a prefix and the namespaceURI is null, if the qualifiedName has a prefix that is "xml" and the namespaceURI is different from "http://www.w3.org/XML/1998/namespace", or if the qualifiedName is "xmlns" and the namespaceURI is different from "http://www.w3.org/2000/xmlns/".

createCDATASection

Creates a CDATASection node whose value is the specified string.

Parameters

data of type DOMString

The data for the CDATASection contents.

Return Value

CDATASection The new CDATASection object.

Exceptions

DOMException NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

createComment

Creates a Comment node given the specified string.

Parameters

data of type DOMString

The data for the node.

Return Value

Comment The new Comment object.

No Exceptions

createDocumentFragment

Creates an empty DocumentFragment object.

Return Value

DocumentFragment A new DocumentFragment.

No Parameters

No Exceptions

createElement

Creates an element of the type specified. Note that the instance returned implements the Element interface, so attributes can be specified directly on the returned object. In addition, if there are known attributes with default values, Attr nodes representing them are automatically created and attached to the element.

To create an element with a qualified name and namespace URI, use the createElementNS method.

Parameters

tagName of type DOMString

The name of the element type to instantiate. For XML, this is case-sensitive. For HTML, the tagName parameter may be provided in any case, but it must be mapped to the canonical uppercase form by the DOM implementation.

Return Value

Element A new Element object with the nodeName attribute set to tagName, and localName, prefix, and namespaceURI set to null.

Exceptions

DOMException INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

createElementNS introduced in DOM Level 2

Creates an element of the given qualified name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString

The namespace URI of the element to create.

qualifiedName of type DOMString

The qualified name of the element type to instantiate.

Return Value

Element A new Element object with the following attributes:

Attribute	Value
Node.nodeName	qualifiedName
Node.namespaceURI	namespaceURI
Node.prefix	prefix, extracted from qualifiedName, or null if there is no prefix
Node.localName	local name, extracted from qualifiedName
Element.tagName	qualifiedName

Exceptions

DOMException INVALID_CHARACTER_ERR: Raised if the specified qualified name contains an illegal character.

NAMESPACE_ERR: Raised if the qualifiedName is malformed, if the qualifiedName has a prefix and the namespaceURI is null, or if the qualifiedName has a prefix that is "xml" and the namespaceURI is different from "http://www.w3.org/XML/1998/namespace" [Namespaces].

createEntityReference

Creates an EntityReference object. In addition, if the referenced entity is known, the child list of the EntityReference node is made the same as that of the corresponding Entity node.

Note: If any descendant of the Entity node has an unbound namespace prefix, the corresponding descendant of the created EntityReference node is also unbound; (its namespaceURI is null). The DOM Level 2 does not support any mechanism to resolve namespace prefixes.

Parameters

name of type DOMString

The name of the entity to reference.

Return Value

EntityReference The new EntityReference object.

Exceptions

DOMException INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

createProcessingInstruction

Creates a ProcessingInstruction node given the specified name and data strings.

Parameters

target of type DOMString

The target part of the processing instruction.

data of type DOMString

The data for the node.

Return Value

ProcessingInstruction The new ProcessingInstruction object.

Exceptions

DOMException INVALID_CHARACTER_ERR: Raised if the specified target contains an illegal character.

NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

createTextNode

Creates a Text node given the specified string.

Parameters

data of type DOMString

The data for the node.

Return Value

Text The new Text object.

No Exceptions

getElementById introduced in DOM Level 2

Returns the Element whose ID is given by elementId. If no such element exists, returns null. Behavior is not defined if more than one element has this ID.

Note: The DOM implementation must have information that says which attributes are of type ID. Attributes with the name "ID" are not of type ID unless so defined. Implementations that do not know whether attributes are of type ID or not are expected to return null.

Parameters

elementId of type DOMString

The unique id value for an element.

Return Value

Element The matching element.

No Exceptions

getElementsByTagName

Returns a NodeList of all the Elements with a given tag name in the order in which they are encountered in a preorder traversal of the Document tree.

Parameters

tagname of type DOMString

The name of the tag to match on. The special value "*" matches all tags.

Return Value

NodeList A new NodeList object containing all the matched Elements.

No Exceptions

getElementsByTagNameNS introduced in DOM Level 2

Returns a NodeList of all the Elements with a given local

name and namespace URI in the order in which they are encountered in a preorder traversal of the Document tree.

Parameters

namespaceURI of type DOMString

The namespace URI of the elements to match on. The special value "*" matches all namespaces.

localName of type DOMString

The local name of the elements to match on. The special value "*" matches all local names.

Return Value

NodeList A new NodeList object containing all the matched Elements.

No Exceptions

importNode introduced in DOM Level 2

Imports a node from another document to this document. The returned node has no parent; (parentNode is null). The source node is not altered or removed from the original document; this method creates a new copy of the source node.

For all nodes, importing a node creates a node object owned by the importing document, with attribute values identical to the source node's nodeName and nodeType, plus the attributes related to namespaces (prefix, localName, and namespaceURI). As in the cloneNode operation on a Node, the source node is not altered.

Additional information is copied as appropriate to the nodeType, attempting to mirror the behavior expected if a fragment of XML or HTML source was copied from one document to another, recognizing that the two documents may have different DTDs in the XML case. The following list describes the specifics for each type of node.

ATTRIBUTE_NODE

The ownerElement attribute is set to null and the specified flag is set to true on the generated Attr. The descendants of the source Attr are recursively imported and the resulting nodes reassembled to form the corresponding subtree.

Note that the deep parameter has no effect on Attr nodes; they always carry their children with them when imported.

DOCUMENT_FRAGMENT_NODE

If the deep option was set to true, the descendants of the source element are recursively imported and the resulting nodes reassembled to form the corresponding subtree. Otherwise, this simply generates an empty DocumentFragment.

DOCUMENT_NODE

Document nodes cannot be imported.

DOCUMENT_TYPE_NODE

DocumentType nodes cannot be imported.

ELEMENT_NODE

Specified attribute nodes of the source element are imported, and the generated Attr nodes are attached to the generated Element. Default attributes are not copied, though if the document being imported into defines default attributes for this element name, those are assigned. If the importNode deep parameter was set to true, the descendants of the source element are recursively imported and the resulting nodes reassembled to form the corresponding subtree.

ENTITY_NODE

Entity nodes can be imported, however in the current

release of the DOM the `DocumentType` is readonly. Ability to add these imported nodes to a `DocumentType` will be considered for addition to a future release of the DOM. On import, the `publicId`, `systemId`, and `notationName` attributes are copied. If a deep import is requested, the descendants of the the source Entity are recursively imported and the resulting nodes reassembled to form the corresponding subtree.

ENTITY_REFERENCE_NODE

Only the `EntityReference` itself is copied, even if a deep import is requested, since the source and destination documents might have defined the entity differently. If the document being imported into provides a definition for this entity name, its value is assigned.

NOTATION_NODE

Notation nodes can be imported, however in the current release of the DOM the `DocumentType` is readonly. Ability to add these imported nodes to a `DocumentType` will be considered for addition to a future release of the DOM. On import, the `publicId` and `systemId` attributes are copied.

Note that the `deep` parameter has no effect on Notation nodes since they never have any children.

PROCESSING_INSTRUCTION_NODE

The imported node copies its target and data values from those of the source node.

TEXT_NODE, CDATA_SECTION_NODE, COMMENT_NODE

These three types of nodes inheriting from `CharacterData` copy their data and length attributes from those of the source node.

Parameters

`importedNode` of type `Node`

The node to import.

`deep` of type `boolean`

If true, recursively import the subtree under the specified node; if false, import only the node itself, as explained above. This has no effect on `Attr`, `EntityReference`, and `Notation` nodes.

Return Value

`Node` The imported node that belongs to this Document.

Exceptions

`DOMException NOT_SUPPORTED_ERR`: Raised if the type of node being imported is not supported.

Interface Node

The `Node` interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree. While all objects implementing the `Node` interface expose methods for dealing with children, not all objects implementing the `Node` interface may have children. For example, `Text` nodes may not have children, and adding children to such nodes results in a `DOMException` being raised.

The attributes `nodeName`, `nodeValue` and `attributes` are included as a mechanism to get at node information without casting down to the specific derived interface. In cases where there is no obvious mapping of these attributes for a specific `nodeType` (e.g., `nodeValue` for an `Element` or `attributes` for a `Comment`), this returns null. Note that the specialized interfaces may contain additional and more convenient mechanisms to get and set the relevant information.

IDL Definition

```

interface Node {

    // NodeType
    const unsigned short      ELEMENT_NODE           = 1;
    const unsigned short      ATTRIBUTE_NODE         = 2;
    const unsigned short      TEXT_NODE              = 3;
    const unsigned short      CDATA_SECTION_NODE     = 4;
    const unsigned short      ENTITY_REFERENCE_NODE  = 5;
    const unsigned short      ENTITY_NODE            = 6;
    const unsigned short      PROCESSING_INSTRUCTION_NODE = 7;
    const unsigned short      COMMENT_NODE           = 8;
    const unsigned short      DOCUMENT_NODE          = 9;
    const unsigned short      DOCUMENT_TYPE_NODE     = 10;
    const unsigned short      DOCUMENT_FRAGMENT_NODE = 11;
    const unsigned short      NOTATION_NODE          = 12;

    readonly attribute DOMString      nodeName;
        attribute DOMString      nodeValue;
        // raises(DOMException) on setting
        // raises(DOMException) on retrieval

    readonly attribute unsigned short .nodeType;
    readonly attribute Node             parentNode;
    readonly attribute NodeList         childNodes;
    readonly attribute Node             firstChild;
    readonly attribute Node             lastChild;
    readonly attribute Node             previousSibling;
    readonly attribute Node             nextSibling;
    readonly attribute NamedNodeMap     attributes;
    // Modified in DOM Level 2:
    readonly attribute Document          ownerDocument;
    Node             insertBefore(in Node newChild,
                                in Node refChild)
                                raises(DOMException);
    Node             replaceChild(in Node newChild,
                                in Node oldChild)
                                raises(DOMException);
    Node             removeChild(in Node oldChild)
                                raises(DOMException);
    Node             appendChild(in Node newChild)
                                raises(DOMException);

    boolean          hasChildNodes();
    Node             cloneNode(in boolean deep);
    // Modified in DOM Level 2:
    void             normalize();
    // Introduced in DOM Level 2:
    boolean          isSupported(in DOMString feature,
                                in DOMString version);

    // Introduced in DOM Level 2:
    readonly attribute DOMString        namespaceURI;
    // Introduced in DOM Level 2:
        attribute DOMString        prefix;
        // raises(DOMException) on setting

    // Introduced in DOM Level 2:
    readonly attribute DOMString        localName;
    // Introduced in DOM Level 2:
    boolean          hasAttributes();
};

```

Definition group NodeType

An integer indicating which type of node this is.

Note: Numeric codes up to 200 are reserved to W3C for possible future use.

Defined Constants

ATTRIBUTE_NODE
The node is an Attr.

CDATA_SECTION_NODE
The node is a CDATASection.

COMMENT_NODE
The node is a Comment.

DOCUMENT_FRAGMENT_NODE
The node is a DocumentFragment.

DOCUMENT_NODE
The node is a Document.

DOCUMENT_TYPE_NODE
The node is a DocumentType.

ELEMENT_NODE
The node is an Element.

ENTITY_NODE
The node is an Entity.

ENTITY_REFERENCE_NODE
The node is an EntityReference.

NOTATION_NODE
The node is a Notation.

PROCESSING_INSTRUCTION_NODE
The node is a ProcessingInstruction.

TEXT_NODE
The node is a Text node.

The values of nodeName, nodeValue, and attributes vary according to the node type as follows:

Interface	nodeName	nodeValue	attributes
Attr	name of attribute	value of attribute	null
CDATASection	#cdata-section	content of the CDATA Section	null
Comment	#comment	content of the comment	null
Document	#document	null	null
DocumentFragment	#document-fragment	null	null
DocumentType	document type name	null	null
Element	tag name	null	NamedNodeMap
Entity	entity name	null	null
EntityReference	name of entity referenced	null	null
Notation	notation name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Text	#text	content of the text node	null

Attributes

`attributes` of type `NamedNodeMap`, readonly

A `NamedNodeMap` containing the attributes of this node (if it is an `Element`) or null otherwise.

`childNodes` of type `NodeList`, readonly

A `NodeList` that contains all children of this node. If there are no children, this is a `NodeList` containing no nodes.

`firstChild` of type `Node`, readonly

The first child of this node. If there is no such node, this returns null.

`lastChild` of type `Node`, readonly

The last child of this node. If there is no such node, this returns null.

`localName` of type `DOMString`, readonly, introduced in DOM Level 2

Returns the local part of the qualified name of this node.

For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `createElement` from the Document interface, this is always null.

`namespaceURI` of type `DOMString`, readonly, introduced in DOM Level 2

The namespace URI of this node, or null if it is unspecified. This is not a computed value that is the result of a namespace lookup based on an examination of the namespace declarations in scope. It is merely the namespace URI given at creation time.

For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `createElement` from the Document interface, this is always null.

Note: Per the Namespaces in XML Specification [Namespaces] an attribute does not inherit its namespace from the element it is attached to. If an attribute is not explicitly given a namespace, it simply has no namespace.

`nextSibling` of type `Node`, readonly

The node immediately following this node. If there is no such node, this returns null.

`nodeName` of type `DOMString`, readonly

The name of this node, depending on its type; see the table above.

`nodeType` of type unsigned short, readonly

A code representing the type of the underlying object, as defined above.

`nodeValue` of type `DOMString`

The value of this node, depending on its type; see the table above. When it is defined to be null, setting it has no effect.

Exceptions on setting

`DOMException NO_MODIFICATION_ALLOWED_ERR`: Raised when the node is readonly.

Exceptions on retrieval

`DOMException DOMSTRING_SIZE_ERR`: Raised when it would return more characters than fit in a `DOMString` variable on the implementation platform.

`ownerDocument` of type `Document`, readonly, modified in DOM Level 2

The `Document` object associated with this node. This is also the `Document` object used to create new nodes. When this node is a `Document` or a `DocumentType` which is not used with any `Document` yet, this is null.

parentNode of type Node, readonly

The parent of this node. All nodes, except Attr, Document, DocumentFragment, Entity, and Notation may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

prefix of type DOMString, introduced in DOM Level 2

The namespace prefix of this node, or null if it is unspecified.

Note that setting this attribute, when permitted, changes the nodeName attribute, which holds the qualified name, as well as the tagName and name attributes of the Element and Attr interfaces, when applicable.

Note also that changing the prefix of an attribute that is known to have a default value, does not make a new attribute with the default value and the original prefix appear, since the namespaceURI and localName do not change.

For nodes of any type other than ELEMENT_NODE and ATTRIBUTE_NODE and nodes created with a DOM Level 1 method, such as createElement from the Document interface, this is always null.

Exceptions on setting

DOMException INVALID_CHARACTER_ERR: Raised if the specified prefix contains an illegal character.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

NAMESPACE_ERR: Raised if the specified prefix is malformed, if the namespaceURI of this node is null, if the specified prefix is "xml" and the namespaceURI of this node is different from "http://www.w3.org/XML/1998/namespace", if this node is an attribute and the specified prefix is "xmlns" and the namespaceURI of this node is different from "http://www.w3.org/2000/xmlns/", or if this node is an attribute and the qualifiedName of this node is "xmlns" [Namespaces].

previousSibling of type Node, readonly

The node immediately preceding this node. If there is no such node, this returns null.

Methods

appendChild

Adds the node newChild to the end of the list of children of this node. If the newChild is already in the tree, it is first removed.

Parameters

newChild of type Node

The node to add.

If it is a DocumentFragment object, the entire contents of the document fragment are moved into the child list of this node

Return Value

Node The node added.

Exceptions

DOMException HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to append is one of this node's ancestors.

WRONG_DOCUMENT_ERR: Raised if newChild was created from a different document than the one that created this node.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

cloneNode

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent; (parentNode is null.).

Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child Text node. Cloning an Attribute directly, as opposed to be cloned as part of an Element cloning operation, returns a specified attribute (specified is true). Cloning any other type of node simply returns a copy of this node.

Note that cloning an immutable subtree results in a mutable copy, but the children of an EntityReference clone are readonly. In addition, clones of unspecified Attr nodes are specified. And, cloning Document, DocumentType, Entity, and Notation nodes is implementation dependent.

Parameters

deep of type boolean

If true, recursively clone the subtree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

Return Value

Node The duplicate node.

No Exceptions

hasAttributes introduced in DOM Level 2

Returns whether this node (if it is an element) has any attributes.

Return Value

boolean true if this node has any attributes, false otherwise.

No Parameters

No Exceptions

hasChildNodes

Returns whether this node has any children.

Return Value

boolean true if this node has any children, false otherwise.

No Parameters

No Exceptions

insertBefore

Inserts the node newChild before the existing child node refChild. If refChild is null, insert newChild at the end of the list of children.

If newChild is a DocumentFragment object, all of its children are inserted, in the same order, before refChild. If the newChild is already in the tree, it is first removed.

Parameters

newChild of type Node

The node to insert.

refChild of type Node

The reference node, i.e., the node before which the new node must be inserted.

Return Value

Node The node being inserted.

Exceptions

DOMException HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to insert is one of this node's ancestors.

WRONG_DOCUMENT_ERR: Raised if newChild was created from a different document than the one that created this node.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly or if the parent of the node being inserted is readonly.

NOT_FOUND_ERR: Raised if refChild is not a child of this node.

isSupported introduced in DOM Level 2

Tests whether the DOM implementation implements a specific feature and that feature is supported by this node.

Parameters

feature of type DOMString

The name of the feature to test. This is the same name which can be passed to the method hasFeature on DOMImplementation.

version of type DOMString

This is the version number of the feature to test. In Level 2, version 1, this is the string "2.0". If the version is not specified, supporting any version of the feature will cause the method to return true.

Return Value

boolean Returns true if the specified feature is supported on this node, false otherwise.

No Exceptions

normalize modified in DOM Level 2

Puts all Text nodes in the full depth of the sub-tree underneath this Node, including attribute nodes, into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates Text nodes, i.e., there are neither adjacent Text nodes nor empty Text nodes. This can be used to ensure that the DOM view of a document is the same as if it were saved and re-loaded, and is useful when operations (such as XPointer [XPointer] lookups) that depend on a particular document tree structure are to be used.

Note: In cases where the document contains CDATASections, the normalize operation alone may not be sufficient, since XPointers do not differentiate between Text nodes and CDATASection nodes.

No Parameters

No Return Value

No Exceptions

removeChild

Removes the child node indicated by oldChild from the list of children, and returns it.

Parameters

oldChild of type Node

The node being removed.

Return Value

Node The node removed.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

NOT_FOUND_ERR: Raised if oldChild is not a child of this node.

replaceChild

Replaces the child node oldChild with newChild in the list of children, and returns the oldChild node.

If newChild is a DocumentFragment object, oldChild is replaced by all of the DocumentFragment children, which are inserted in the same order. If the newChild is already in the tree, it is first removed.

Parameters

newChild of type Node

The new node to put in the child list.

oldChild of type Node

The node being replaced in the list.

Return Value

Node The node replaced.

Exceptions

DOMException HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to put in is one of this node's ancestors.

WRONG_DOCUMENT_ERR: Raised if newChild was created from a different document than the one that created this node.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node or the parent of the new node is readonly.

NOT_FOUND_ERR: Raised if oldChild is not a child of this node.

Interface NodeList

The NodeList interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented. NodeList objects in the DOM are live.

The items in the NodeList are accessible via an integral index, starting from 0.

IDL Definition

```
interface NodeList {  
    Node          item(unsigned long index);  
    readonly attribute unsigned long    length;  
};
```

Attributes

length of type unsigned long, readonly

The number of nodes in the list. The range of valid child node indices is 0 to length-1 inclusive.

Methods

item

Returns the indexth item in the collection. If index is greater than or equal to the number of nodes in the list, this returns null.

Parameters

index of type unsigned long

Index into the collection.

Return Value

Node The node at the indexth position in the NodeList, or null if that is not a valid index.

No Exceptions

Interface NamedNodeMap

Objects implementing the NamedNodeMap interface are used to represent collections of nodes that can be accessed by name. Note that NamedNodeMap does not inherit from NodeList; NamedNodeMaps are not maintained in any particular order. Objects contained in an object implementing NamedNodeMap may also be accessed by an ordinal index, but this is simply to allow convenient enumeration of the contents of a NamedNodeMap, and does not imply that the DOM specifies an order to these Nodes.

NamedNodeMap objects in the DOM are live.

IDL Definition

```
interface NamedNodeMap {
    Node          getNamedItem(in DOMString name);
    Node          setNamedItem(in Node arg)
                                   raises(DOMException);
    Node          removeNamedItem(in DOMString name)
                                   raises(DOMException);
    Node          item(in unsigned long index);
    readonly attribute unsigned long    length;
    // Introduced in DOM Level 2:
    Node          getNamedItemNS(in DOMString namespaceURI,
                                   in DOMString localName);
    // Introduced in DOM Level 2:
    Node          setNamedItemNS(in Node arg)
                                   raises(DOMException);
    // Introduced in DOM Level 2:
    Node          removeNamedItemNS(in DOMString namespaceURI,
                                   in DOMString localName)
                                   raises(DOMException);
};
```

Attributes

length of type unsigned long, readonly

The number of nodes in this map. The range of valid child node indices is 0 to length-1 inclusive.

Methods

getNamedItem

Retrieves a node specified by name.

Parameters

name of type DOMString

The nodeName of a node to retrieve.

Return Value

Node A Node (of any type) with the specified nodeName, or null if it does not identify any node in this map.

No Exceptions

getNamedItemNS introduced in DOM Level 2

Retrieves a node specified by local name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString

The namespace URI of the node to retrieve.
localName of type DOMString
The local name of the node to retrieve.
Return Value
Node A Node (of any type) with the specified local name and namespace URI, or null if they do not identify any node in this map.
No Exceptions

item

Returns the indexth item in the map. If index is greater than or equal to the number of nodes in this map, this returns null.
Parameters
index of type unsigned long
Index into this map.
Return Value
Node The node at the indexth position in the map, or null if that is not a valid index.
No Exceptions

removeNamedItem

Removes a node specified by name. When this map contains the attributes attached to an element, if the removed attribute is known to have a default value, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.
Parameters
name of type DOMString
The nodeName of the node to remove.
Return Value
Node The node removed from this map if a node with such a name exists.
Exceptions
DOMException NOT_FOUND_ERR: Raised if there is no node named name in this map.

NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

removeNamedItemNS introduced in DOM Level 2

Removes a node specified by local name and namespace URI. A removed attribute may be known to have a default value when this map contains the attributes attached to an element, as returned by the attributes attribute of the Node interface. If so, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.
HTML-only DOM implementations do not need to implement this method.
Parameters
namespaceURI of type DOMString
The namespace URI of the node to remove.
localName of type DOMString
The local name of the node to remove.
Return Value
Node The node removed from this map if a node with such a local name and namespace URI exists.
Exceptions
DOMException NOT_FOUND_ERR: Raised if there is no node with the specified namespaceURI and localName in this map.

NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

setNamedItem

Adds a node using its nodeName attribute. If a node with that name is already present in this map, it is replaced by the new one.

As the nodeName attribute is used to derive the name which the node must be stored under, multiple nodes of certain types (those that have a "special" string value) cannot be stored as the names would clash. This is seen as preferable to allowing nodes to be aliased.

Parameters

arg of type Node

A node to store in this map. The node will later be accessible using the value of its nodeName attribute.

Return Value

Node If the new Node replaces an existing node the replaced Node is returned, otherwise null is returned.

Exceptions

DOMException WRONG_DOCUMENT_ERR: Raised if arg was created from a different document than the one that created this map.

NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

INUSE_ATTRIBUTE_ERR: Raised if arg is an Attr that is already an attribute of another Element object. The DOM user must explicitly clone Attr nodes to re-use them in other elements.

setNamedItemNS introduced in DOM Level 2

Adds a node using its namespaceURI and localName. If a node with that namespace URI and that local name is already present in this map, it is replaced by the new one.

HTML-only DOM implementations do not need to implement this method.

Parameters

arg of type Node

A node to store in this map. The node will later be accessible using the value of its namespaceURI and localName attributes.

Return Value

Node If the new Node replaces an existing node the replaced Node is returned, otherwise null is returned.

Exceptions

DOMException WRONG_DOCUMENT_ERR: Raised if arg was created from a different document than the one that created this map.

NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

INUSE_ATTRIBUTE_ERR: Raised if arg is an Attr that is already an attribute of another Element object. The DOM user must explicitly clone Attr nodes to re-use them in other elements.

Interface CharacterData

The CharacterData interface extends Node with a set of attributes and methods for accessing character data in the DOM. For clarity this set is defined here rather than on each object that uses these attributes

and methods. No DOM objects correspond directly to `CharacterData`, though `Text` and others do inherit the interface from it. All offsets in this interface start from 0.

As explained in the `DOMString` interface, text strings in the DOM are represented in UTF-16, i.e. as a sequence of 16-bit units. In the following, the term 16-bit units is used whenever necessary to indicate that indexing on `CharacterData` is done in 16-bit units.

IDL Definition

```
interface CharacterData : Node {
    attribute DOMString          data;
                                   // raises(DOMException) on setting
                                   // raises(DOMException) on retrieval

    readonly attribute unsigned long    length;
    DOMString          substringData(in unsigned long offset,
                                     in unsigned long count)
                                   raises(DOMException);
    void              appendData(in DOMString arg)
                                   raises(DOMException);
    void              insertData(in unsigned long offset,
                                in DOMString arg)
                                   raises(DOMException);
    void              deleteData(in unsigned long offset,
                                in unsigned long count)
                                   raises(DOMException);
    void              replaceData(in unsigned long offset,
                                in unsigned long count,
                                in DOMString arg)
                                   raises(DOMException);
};
```

Attributes

`data` of type `DOMString`

The character data of the node that implements this interface. The DOM implementation may not put arbitrary limits on the amount of data that may be stored in a `CharacterData` node. However, implementation limits may mean that the entirety of a node's data may not fit into a single `DOMString`. In such cases, the user may call `substringData` to retrieve the data in appropriately sized pieces.

Exceptions on setting

`DOMException NO_MODIFICATION_ALLOWED_ERR`: Raised when the node is readonly.

Exceptions on retrieval

`DOMException DOMSTRING_SIZE_ERR`: Raised when it would return more characters than fit in a `DOMString` variable on the implementation platform.

`length` of type `unsigned long`, readonly

The number of 16-bit units that are available through `data` and the `substringData` method below. This may have the value zero, i.e., `CharacterData` nodes may be empty.

Methods

`appendData`

Append the string to the end of the character data of the node. Upon success, `data` provides access to the concatenation of data and the `DOMString` specified.

Parameters

`arg` of type `DOMString`

The DOMString to append.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

deleteData

Remove a range of 16-bit units from the node. Upon success, data and length reflect the change.

Parameters

offset of type unsigned long

The offset from which to start removing.

count of type unsigned long

The number of 16-bit units to delete. If the sum of offset and count exceeds length then all 16-bit units from offset to the end of the data are deleted.

Exceptions

DOMException INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

insertData

Insert a string at the specified 16-bit unit offset.

Parameters

offset of type unsigned long

The character offset at which to insert.

arg of type DOMString

The DOMString to insert.

Exceptions

DOMException INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

replaceData

Replace the characters starting at the specified 16-bit unit offset with the specified string.

Parameters

offset of type unsigned long

The offset from which to start replacing.

count of type unsigned long

The number of 16-bit units to replace. If the sum of offset and count exceeds length, then all 16-bit units to the end of the data are replaced; (i.e., the effect is the same as a remove method call with the same range, followed by an append method invocation).

arg of type DOMString

The DOMString with which the range must be replaced.

Exceptions

DOMException INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

substringData

Extracts a range of data from the node.

Parameters

offset of type unsigned long

Start offset of substring to extract.

count of type unsigned long

The number of 16-bit units to extract.

Return Value

DOMString The specified substring. If the sum of offset and count exceeds the length, then all 16-bit units to the end of the data are returned.

Exceptions

DOMException INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

DOMSTRING_SIZE_ERR: Raised if the specified range of text does not fit into a DOMString.

Interface Attr

The Attr interface represents an attribute in an Element object. Typically the allowable values for the attribute are defined in a document type definition.

Attr objects inherit the Node interface, but since they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree. Thus, the Node attributes parentNode, previousSibling, and nextSibling have a null value for Attr objects. The DOM takes the view that attributes are properties of elements rather than having a separate identity from the elements they are associated with; this should make it more efficient to implement such features as default attributes associated with all elements of a given type. Furthermore, Attr nodes may not be immediate children of a DocumentFragment. However, they can be associated with Element nodes contained within a DocumentFragment. In short, users and implementors of the DOM need to be aware that Attr nodes have some things in common with other objects inheriting the Node interface, but they also are quite distinct.

The attribute's effective value is determined as follows: if this attribute has been explicitly assigned any value, that value is the attribute's effective value; otherwise, if there is a declaration for this attribute, and that declaration includes a default value, then that default value is the attribute's effective value; otherwise, the attribute does not exist on this element in the structure model until it has been explicitly added. Note that the nodeValue attribute on the Attr instance can also be used to retrieve the string version of the attribute's value(s).

In XML, where the value of an attribute can contain entity references, the child nodes of the Attr node may be either Text or EntityReference nodes (when these are in use; see the description of EntityReference for discussion). Because the DOM Core is not aware of attribute types, it treats all attribute values as simple strings, even if the DTD or schema declares them as having tokenized types.

IDL Definition

```
interface Attr : Node {
```

```
    readonly attribute DOMString    name;
    readonly attribute boolean      specified;
        attribute DOMString        value;
        // raises(DOMException) on setting

    // Introduced in DOM Level 2:
    readonly attribute Element      ownerElement;
};
```

Attributes

name of type `DOMString`, **readonly**
Returns the name of this attribute.

ownerElement of type `Element`, **readonly**, introduced in DOM Level 2
The `Element` node this attribute is attached to or null if this attribute is not in use.

specified of type `boolean`, **readonly**
If this attribute was explicitly given a value in the original document, this is true; otherwise, it is false. Note that the implementation is in charge of this attribute, not the user. If the user changes the value of the attribute (even if it ends up having the same value as the default value) then the **specified** flag is automatically flipped to true. To re-specify the attribute as the default value from the DTD, the user must delete the attribute. The implementation will then make a new attribute available with **specified** set to false and the default value (if one exists).
In summary:

- + If the attribute has an assigned value in the document then **specified** is true, and the value is the assigned value.
- + If the attribute has no assigned value in the document and has a default value in the DTD, then **specified** is false, and the value is the default value in the DTD.
- + If the attribute has no assigned value in the document and has a value of `#IMPLIED` in the DTD, then the attribute does not appear in the structure model of the document.
- + If the **ownerElement** attribute is null (i.e. because it was just created or was set to null by the various removal and cloning operations) **specified** is true.

value of type `DOMString`
On retrieval, the value of the attribute is returned as a string. Character and general entity references are replaced with their values. See also the method `getAttribute` on the `Element` interface.
On setting, this creates a `Text` node with the unparsed contents of the string. I.e. any characters that an XML processor would recognize as markup are instead treated as literal text. See also the method `setAttribute` on the `Element` interface.
Exceptions on setting
`DOMException NO_MODIFICATION_ALLOWED_ERR`: Raised when the node is **readonly**.

Interface `Element`

The `Element` interface represents an element in an HTML or XML document. Elements may have attributes associated with them; since the `Element` interface inherits from `Node`, the generic `Node` interface attribute `attributes` may be used to retrieve the set of all attributes for an element. There are methods on the `Element` interface to retrieve either

an Attr object by name or an attribute value by name. In XML, where an attribute value may contain entity references, an Attr object should be retrieved to examine the possibly fairly complex sub-tree representing the attribute value. On the other hand, in HTML, where all attributes have simple string values, methods to directly access an attribute value can safely be used as a convenience.

Note: In DOM Level 2, the method `normalize` is inherited from the Node interface where it was moved.

IDL Definition

```
interface Element : Node {
    readonly attribute DOMString      tagName;
    DOMString      getAttribute(in DOMString name);
    void           setAttribute(in DOMString name,
                               in DOMString value)
                               raises(DOMException);
    void           removeAttribute(in DOMString name)
                               raises(DOMException);
    Attr           getAttributeNode(in DOMString name);
    Attr           setAttributeNode(in Attr newAttr)
                               raises(DOMException);
    Attr           removeAttributeNode(in Attr oldAttr)
                               raises(DOMException);
    NodeList       getElementsByTagName(in DOMString name);
    // Introduced in DOM Level 2:
    DOMString      getAttributeNS(in DOMString namespaceURI,
                                  in DOMString localName);
    // Introduced in DOM Level 2:
    void           setAttributeNS(in DOMString namespaceURI,
                                  in DOMString qualifiedName,
                                  in DOMString value)
                                  raises(DOMException);
    // Introduced in DOM Level 2:
    void           removeAttributeNS(in DOMString namespaceURI,
                                     in DOMString localName)
                                     raises(DOMException);
    // Introduced in DOM Level 2:
    Attr           getAttributeNodeNS(in DOMString namespaceURI,
                                       in DOMString localName);
    // Introduced in DOM Level 2:
    Attr           setAttributeNodeNS(in Attr newAttr)
                                       raises(DOMException);
    // Introduced in DOM Level 2:
    NodeList       getElementsByTagNameNS(in DOMString namespaceURI,
                                          in DOMString localName);
    // Introduced in DOM Level 2:
    boolean        hasAttribute(in DOMString name);
    // Introduced in DOM Level 2:
    boolean        hasAttributeNS(in DOMString namespaceURI,
                                  in DOMString localName);
};
```

Attributes

`tagName` of type `DOMString`, `readonly`
The name of the element. For example, in:

```
<elementExample id="demo">
    ...
</elementExample> ,
```

tagName has the value "elementExample". Note that this is case-preserving in XML, as are all of the operations of the DOM. The HTML DOM returns the tagName of an HTML element in the canonical uppercase form, regardless of the case in the source HTML document.

Methods

getAttribute

Retrieves an attribute value by name.

Parameters

name of type DOMString

The name of the attribute to retrieve.

Return Value

DOMString The Attr value as a string, or the empty string if that attribute does not have a specified or default value.

No Exceptions

getAttributeNS introduced in DOM Level 2

Retrieves an attribute value by local name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString

The namespace URI of the attribute to retrieve.

localName of type DOMString

The local name of the attribute to retrieve.

Return Value

DOMString The Attr value as a string, or the empty string if that attribute does not have a specified or default value.

No Exceptions

getAttributeNode

Retrieves an attribute node by name.

To retrieve an attribute node by qualified name and namespace URI, use the getAttributeNodeNS method.

Parameters

name of type DOMString

The name (nodeName) of the attribute to retrieve.

Return Value

Attr The Attr node with the specified name (nodeName) or null if there is no such attribute.

No Exceptions

getAttributeNodeNS introduced in DOM Level 2

Retrieves an Attr node by local name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString

The namespace URI of the attribute to retrieve.

localName of type DOMString

The local name of the attribute to retrieve.

Return Value

Attr The Attr node with the specified attribute local name and namespace URI or null if there is no such attribute.

No Exceptions

getElementsByTagName

Returns a NodeList of all descendant Elements with a given tag name, in the order in which they are encountered in a preorder traversal of this Element tree.

Parameters

name of type DOMString

The name of the tag to match on. The special value "*" matches all tags.

Return Value

NodeList A list of matching Element nodes.

No Exceptions

getElementsByTagNameNS introduced in DOM Level 2

Returns a NodeList of all the descendant Elements with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of this Element tree. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString

The namespace URI of the elements to match on. The special value "*" matches all namespaces.

localName of type DOMString

The local name of the elements to match on. The special value "*" matches all local names.

Return Value

NodeList A new NodeList object containing all the matched Elements.

No Exceptions

hasAttribute introduced in DOM Level 2

Returns true when an attribute with a given name is specified on this element or has a default value, false otherwise.

Parameters

name of type DOMString

The name of the attribute to look for.

Return Value

boolean true if an attribute with the given name is specified on this element or has a default value, false otherwise.

No Exceptions

hasAttributeNS introduced in DOM Level 2

Returns true when an attribute with a given local name and namespace URI is specified on this element or has a default value, false otherwise. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString

The namespace URI of the attribute to look for.

localName of type DOMString

The local name of the attribute to look for.

Return Value

boolean true if an attribute with the given local name and namespace URI is specified or has a default value on this element, false otherwise.

No Exceptions

removeAttribute

Removes an attribute by name. If the removed attribute is known to have a default value, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

To remove an attribute by local name and namespace URI, use the removeAttributeNS method.

Parameters

name of type DOMString

The name of the attribute to remove.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

removeAttributeNS introduced in DOM Level 2

Removes an attribute by local name and namespace URI. If the removed attribute has a default value it is immediately replaced. The replacing attribute has the same namespace URI and local name, as well as the original prefix. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString

The namespace URI of the attribute to remove.

localName of type DOMString

The local name of the attribute to remove.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

removeAttributeNode

Removes the specified attribute node. If the removed Attr has a default value it is immediately replaced. The replacing attribute has the same namespace URI and local name, as well as the original prefix, when applicable.

Parameters

oldAttr of type Attr

The Attr node to remove from the attribute list.

Return Value

Attr The Attr node that was removed.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

NOT_FOUND_ERR: Raised if oldAttr is not an attribute of the element.

setAttribute

Adds a new attribute. If an attribute with that name is already present in the element, its value is changed to be that of the value parameter. This value is a simple string; it is not parsed as it is being set. So any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately escaped by the implementation when it is written out. In order to assign an attribute value that contains entity references, the user must create an Attr node plus any Text and EntityReference nodes, build the appropriate subtree, and use setAttributeNode to assign it as the value of an attribute. To set an attribute with a qualified name and namespace URI, use the setAttributeNS method.

Parameters

name of type DOMString

The name of the attribute to create or alter.

value of type DOMString

Value to set in string form.

Exceptions

DOMException INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

setAttributeNS introduced in DOM Level 2

Adds a new attribute. If an attribute with the same local name and namespace URI is already present on the element, its prefix is changed to be the prefix part of the qualifiedName, and its value is changed to be the value parameter. This value is a simple string; it is not parsed as it is being set. So any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately escaped by the implementation when it is written out. In order to assign an attribute value that contains entity references, the user must create an Attr node plus any Text and EntityReference nodes, build the appropriate subtree, and use setAttributeNodeNS or setAttributeNode to assign it as the value of an attribute. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString

The namespace URI of the attribute to create or alter.

qualifiedName of type DOMString

The qualified name of the attribute to create or alter.

value of type DOMString

The value to set in string form.

Exceptions

DOMException INVALID_CHARACTER_ERR: Raised if the specified qualified name contains an illegal character.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

NAMESPACE_ERR: Raised if the qualifiedName is malformed, if the qualifiedName has a prefix and the namespaceURI is null, if the qualifiedName has a prefix that is "xml" and the namespaceURI is different from "http://www.w3.org/XML/1998/namespace", or if the qualifiedName is "xmlns" and the namespaceURI is different from "http://www.w3.org/2000/xmlns/".

No Return Value

setAttributeNode

Adds a new attribute node. If an attribute with that name (nodeName) is already present in the element, it is replaced by the new one.

To add a new attribute node with a qualified name and namespace URI, use the setAttributeNodeNS method.

Parameters

newAttr of type Attr

The Attr node to add to the attribute list.

Return Value

Attr If the newAttr attribute replaces an existing attribute, the replaced Attr node is returned, otherwise null is returned.

Exceptions

DOMException WRONG_DOCUMENT_ERR: Raised if newAttr was created from a different document than the one that created the element.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

INUSE_ATTRIBUTE_ERR: Raised if newAttr is already an attribute of another Element object. The DOM user must explicitly clone Attr nodes to re-use them in other elements.

setAttributeNodeNS introduced in DOM Level 2

Adds a new attribute. If an attribute with that local name and that namespace URI is already present in the element, it is replaced by the new one.

HTML-only DOM implementations do not need to implement this method.

Parameters

newAttr of type Attr

The Attr node to add to the attribute list.

Return Value

Attr If the newAttr attribute replaces an existing attribute with the same local name and namespace URI, the replaced Attr node is returned, otherwise null is returned.

Exceptions

DOMException WRONG_DOCUMENT_ERR: Raised if newAttr was created from a different document than the one that created the element.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

INUSE_ATTRIBUTE_ERR: Raised if newAttr is already an attribute of another Element object. The DOM user must explicitly clone Attr nodes to re-use them in other elements.

Interface Text

The Text interface inherits from CharacterData and represents the textual content (termed character data in XML) of an Element or Attr. If there is no markup inside an element's content, the text is contained in a single object implementing the Text interface that is the only child of the element. If there is markup, it is parsed into the information items (elements, comments, etc.) and Text nodes that form the list of children of the element.

When a document is first made available via the DOM, there is only one Text node for each block of text. Users may create adjacent Text nodes that represent the contents of a given element without any intervening markup, but should be aware that there is no way to represent the separations between these nodes in XML or HTML, so they will not (in general) persist between DOM editing sessions. The normalize() method on Node merges any such adjacent Text objects into a single node for each block of text.

IDL Definition

```
interface Text : CharacterData {  
    Text                splitText(in unsigned long offset)  
                                raises(DOMException);  
};
```

Methods

splitText

Breaks this node into two nodes at the specified offset, keeping both in the tree as siblings. After being split, this node will contain all the content up to the offset point. A

new node of the same type, which contains all the content at and after the offset point, is returned. If the original node had a parent node, the new node is inserted as the next sibling of the original node. When the offset is equal to the length of this node, the new node has no data.

Parameters

offset of type unsigned long

The 16-bit unit offset at which to split, starting from 0.

Return Value

Text The new node, of the same type as this node.

Exceptions

DOMException INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

Interface Comment

This interface inherits from `CharacterData` and represents the content of a comment, i.e., all the characters between the starting '`<!--`' and ending '`-->`'. Note that this is the definition of a comment in XML, and, in practice, HTML, although some HTML tools may implement the full SGML comment structure.

IDL Definition

```
interface Comment : CharacterData {  
};
```

1.3. Extended Interfaces

The interfaces defined here form part of the DOM Core specification, but objects that expose these interfaces will never be encountered in a DOM implementation that deals only with HTML. As such, HTML-only DOM implementations [DOM Level 2 HTML] do not need to have objects that implement these interfaces.

The interfaces found within this section are not mandatory. A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "XML" and "2.0" (respectively) to determine whether or not this module is supported by the implementation. In order to fully support this module, an implementation must also support the "Core" feature defined in Fundamental Interfaces. Please refer to additional information about Conformance in this specification.

Interface CDATASection

CDATA sections are used to escape blocks of text containing characters that would otherwise be regarded as markup. The only delimiter that is recognized in a CDATA section is the "`]]>`" string that ends the CDATA section. CDATA sections cannot be nested. Their primary purpose is for including material such as XML fragments, without needing to escape all the delimiters.

The `DOMString` attribute of the `Text` node holds the text that is contained by the CDATA section. Note that this may contain characters that need to be escaped outside of CDATA sections and that, depending on the character encoding ("charset") chosen for serialization, it may

be impossible to write out some characters as part of a CDATA section.

The CDATASection interface inherits from the CharacterData interface through the Text interface. Adjacent CDATASection nodes are not merged by use of the normalize method of the Node interface.

Note: Because no markup is recognized within a CDATASection, character numeric references cannot be used as an escape mechanism when serializing. Therefore, action needs to be taken when serializing a CDATASection with a character encoding where some of the contained characters cannot be represented. Failure to do so would not produce well-formed XML.

One potential solution in the serialization process is to end the CDATA section before the character, output the character using a character reference or entity reference, and open a new CDATA section for any further characters in the text node. Note, however, that some code conversion libraries at the time of writing do not return an error or exception when a character is missing from the encoding, making the task of ensuring that data is not corrupted on serialization more difficult.

IDL Definition

```
interface CDATASection : Text {  
};
```

Interface DocumentType

Each Document has a doctype attribute whose value is either null or a DocumentType object. The DocumentType interface in the DOM Core provides an interface to the list of entities that are defined for the document, and little else because the effect of namespaces and the various XML schema efforts on DTD representation are not clearly understood as of this writing.

The DOM Level 2 doesn't support editing DocumentType nodes.

IDL Definition

```
interface DocumentType : Node {  
    readonly attribute DOMString      name;  
    readonly attribute NamedNodeMap    entities;  
    readonly attribute NamedNodeMap    notations;  
    // Introduced in DOM Level 2:  
    readonly attribute DOMString      publicId;  
    // Introduced in DOM Level 2:  
    readonly attribute DOMString      systemId;  
    // Introduced in DOM Level 2:  
    readonly attribute DOMString      internalSubset;  
};
```

Attributes

entities of type NamedNodeMap, readonly

A NamedNodeMap containing the general entities, both external and internal, declared in the DTD. Parameter entities are not contained. Duplicates are discarded. For example in:

```
<!DOCTYPE ex SYSTEM "ex.dtd" [  
    <!ENTITY foo "foo">  
    <!ENTITY bar "bar">  
    <!ENTITY bar "bar2">
```



```
<!ENTITY % baz "baz">
]>
<ex/>
```

the interface provides access to foo and the first declaration of bar but not the second declaration of bar or baz. Every node in this map also implements the Entity interface.

The DOM Level 2 does not support editing entities, therefore entities cannot be altered in any way.

internalSubset of type DOMString, readonly, introduced in DOM Level 2

The internal subset as a string.

Note: The actual content returned depends on how much information is available to the implementation. This may vary depending on various parameters, including the XML processor used to build the document.

name of type DOMString, readonly

The name of DTD; i.e., the name immediately following the DOCTYPE keyword.

notations of type NamedNodeMap, readonly

A NamedNodeMap containing the notations declared in the DTD. Duplicates are discarded. Every node in this map also implements the Notation interface.

The DOM Level 2 does not support editing notations, therefore notations cannot be altered in any way.

publicId of type DOMString, readonly, introduced in DOM Level 2

The public identifier of the external subset.

systemId of type DOMString, readonly, introduced in DOM Level 2

The system identifier of the external subset.

Interface Notation

This interface represents a notation declared in the DTD. A notation either declares, by name, the format of an unparsed entity (see section 4.7 of the XML 1.0 specification [XML]), or is used for formal declaration of processing instruction targets (see section 2.6 of the XML 1.0 specification [XML]). The nodeName attribute inherited from Node is set to the declared name of the notation.

The DOM Level 1 does not support editing Notation nodes; they are therefore readonly.

A Notation node does not have any parent.

IDL Definition

```
interface Notation : Node {
    readonly attribute DOMString    publicId;
    readonly attribute DOMString    systemId;
};
```

Attributes

publicId of type DOMString, readonly

The public identifier of this notation. If the public identifier was not specified, this is null.

systemId of type DOMString, readonly

The system identifier of this notation. If the system identifier was not specified, this is null.

Interface Entity

This interface represents an entity, either parsed or unparsed, in an XML document. Note that this models the entity itself not the entity declaration. Entity declaration modeling has been left for a later Level of the DOM specification.

The nodeName attribute that is inherited from Node contains the name of the entity.

An XML processor may choose to completely expand entities before the structure model is passed to the DOM; in this case there will be no EntityReference nodes in the document tree.

XML does not mandate that a non-validating XML processor read and process entity declarations made in the external subset or declared in external parameter entities. This means that parsed entities declared in the external subset need not be expanded by some classes of applications, and that the replacement value of the entity may not be available. When the replacement value is available, the corresponding Entity node's child list represents the structure of that replacement text. Otherwise, the child list is empty.

The DOM Level 2 does not support editing Entity nodes; if a user wants to make changes to the contents of an Entity, every related EntityReference node has to be replaced in the structure model by a clone of the Entity's contents, and then the desired changes must be made to each of those clones instead. Entity nodes and all their descendants are readonly.

An Entity node does not have any parent.

Note: If the entity contains an unbound namespace prefix, the namespaceURI of the corresponding node in the Entity node subtree is null. The same is true for EntityReference nodes that refer to this entity, when they are created using the createEntityReference method of the Document interface. The DOM Level 2 does not support any mechanism to resolve namespace prefixes.

IDL Definition

```
interface Entity : Node {
    readonly attribute DOMString      publicId;
    readonly attribute DOMString      systemId;
    readonly attribute DOMString      notationName;
};
```

Attributes

notationName of type DOMString, readonly
For unparsed entities, the name of the notation for the entity. For parsed entities, this is null.

publicId of type DOMString, readonly
The public identifier associated with the entity, if specified. If the public identifier was not specified, this is null.

systemId of type DOMString, readonly
The system identifier associated with the entity, if specified. If the system identifier was not specified, this is null.

Interface EntityReference

EntityReference objects may be inserted into the structure model when

an entity reference is in the source document, or when the user wishes to insert an entity reference. Note that character references and references to predefined entities are considered to be expanded by the HTML or XML processor so that characters are represented by their Unicode equivalent rather than by an entity reference. Moreover, the XML processor may completely expand references to entities while building the structure model, instead of providing `EntityReference` objects. If it does provide such objects, then for a given `EntityReference` node, it may be that there is no `Entity` node representing the referenced entity. If such an `Entity` exists, then the subtree of the `EntityReference` node is in general a copy of the `Entity` node subtree. However, this may not be true when an entity contains an unbound namespace prefix. In such a case, because the namespace prefix resolution depends on where the entity reference is, the descendants of the `EntityReference` node may be bound to different namespace URIs.

As for `Entity` nodes, `EntityReference` nodes and all their descendants are readonly.

IDL Definition

```
interface EntityReference : Node {  
};
```

Interface `ProcessingInstruction`

The `ProcessingInstruction` interface represents a "processing instruction", used in XML as a way to keep processor-specific information in the text of the document.

IDL Definition

```
interface ProcessingInstruction : Node {  
    readonly attribute DOMString    target;  
    attribute DOMString             data;  
                                     // raises(DOMException) on setting  
};
```

Attributes

data of type `DOMString`
The content of this processing instruction. This is from the first non white space character after the target to the character immediately preceding the `?>`.
Exceptions on setting
`DOMException NO_MODIFICATION_ALLOWED_ERR`: Raised when the node is readonly.

target of type `DOMString`, readonly
The target of this processing instruction. XML defines this as being the first token following the markup that begins the processing instruction.

13 November, 2000

Appendix A: Changes

Editors

Arnaud Le Hors, IBM
Philippe Le Hégaré, W3C

A.1: Changes between DOM Level 1 Core and DOM Level 2 Core

OMG IDL

The DOM Level 2 specifications are now using Corba 2.3.1 instead of Corba 2.2.

Type DOMString

The definition of DOMString in IDL is now a valuetype.

A.1.1: Changes to DOM Level 1 Core interfaces and exceptions

Interface Attr

The Attr interface has one new attribute: ownerElement.

Interface Document

The Document interface has five new methods: importNode, createElementNS, createAttributeNS, getElementsByTagNameNS and getElementById.

Interface NamedNodeMap

The NamedNodeMap interface has three new methods: getNamedItemNS, setNamedItemNS, removeNamedItemNS.

Interface Node

The Node interface has two new methods: isSupported and hasAttributes. normalize, previously in the Element interface, has been moved in the Node interface.

The Node interface has three new attributes: namespaceURI, prefix and localName.

The ownerDocument attribute was specified to be null when the node is a Document. It now is also null when the node is a DocumentType which is not used with any Document yet.

Interface DocumentType

The DocumentType interface has three attributes: publicId, systemId and internalSubset.

Interface DOMImplementation

The DOMImplementation interface has two new methods: createDocumentType and createDocument.

Interface Element

The Element interface has eight new methods: getAttributeNS, setAttributeNS, removeAttributeNS, getAttributeNodeNS, setAttributeNodeNS, getElementsByTagNameNS, hasAttribute and hasAttributeNS.

The method normalize is now inherited from the Node interface where it was moved.

Exception DOMException

The DOMException has five new exception codes: INVALID_STATE_ERR, SYNTAX_ERR, INVALID_MODIFICATION_ERR, NAMESPACE_ERR and INVALID_ACCESS_ERR.

A.1.2: New features

A.1.2.1: New types

DOMTimeStamp

The DOMTimeStamp type was added to the Core module.

13 November, 2000

Appendix B: Accessing code point boundaries

Mark Davis, IBM

Lauren Wood, SoftQuad Software Inc.

Table of contents

- * 2.1. Introduction
- * 2.2. Methods
 - o StringExtend

B.1: Introduction

This appendix is an informative, not a normative, part of the Level 2 DOM specification.

Characters are represented in Unicode by numbers called code points (also called scalar values). These numbers can range from 0 up to 1,114,111 = 10FFFF₁₆ (although some of these values are illegal). Each code point can be directly encoded with a 32-bit code unit. This encoding is termed UCS-4 (or UTF-32). The DOM specification, however, uses UTF-16, in which the most frequent characters (which have values less than FFFF₁₆) are represented by a single 16-bit code unit, while characters above FFFF₁₆ use a special pair of code units called a surrogate pair. For more information, see [Unicode] or the Unicode Web site.

While indexing by code points as opposed to code units is not common in programs, some specifications such as XPath (and therefore XSLT and XPointer) use code point indices. For interfacing with such formats it is recommended that the programming language provide string processing methods for converting code point indices to code unit indices and back. Some languages do not provide these functions natively; for these it is recommended that the native String type that is bound to DOMString be extended to enable this conversion. An example of how such an API might look is supplied below.

Note: Since these methods are supplied as an illustrative example of the type of functionality that is required, the names of the methods, exceptions, and interface may differ from those given here.

B.2: Methods

Interface StringExtend

Extensions to a language's native String class or interface

IDL Definition

```
interface StringExtend {  
    int findOffset16(in int offset32)
```

```
raises(StringIndexOutOfBoundsException);  
    int findOffset32(in int offset16)
```

```
raises(StringIndexOutOfBoundsException);  
};
```

Methods

findOffset16

Returns the UTF-16 offset that corresponds to a UTF-32 offset. Used for random access.

Note: You can always round-trip from a UTF-32 offset to a UTF-16 offset and back. You can round-trip from a UTF-16 offset to a UTF-32 offset and back if and only if the offset16 is not in the middle of a surrogate pair. Unmatched surrogates count as a single UTF-16 value.

Parameters
offset32 of type int
UTF-32 offset.
Return Value
int UTF-16 offset
Exceptions
StringIndexOutOfBoundsException if offset32 is out of bounds.

findOffset32

Returns the UTF-32 offset corresponding to a UTF-16 offset. Used for random access. To find the UTF-32 length of a string, use:

```
len32 = findOffset32(source, source.length());
```

Note: If the UTF-16 offset is into the middle of a surrogate pair, then the UTF-32 offset of the end of the pair is returned; that is, the index of the char after the end of the pair. You can always round-trip from a UTF-32 offset to a UTF-16 offset and back. You can round-trip from a UTF-16 offset to a UTF-32 offset and back if and only if the offset16 is not in the middle of a surrogate pair. Unmatched surrogates count as a single UTF-16 value.

Parameters
offset16 of type int
UTF-16 offset
Return Value
int UTF-32 offset
Exceptions
StringIndexOutOfBoundsException if offset16 is out of bounds.

13 November, 2000

Appendix C: IDL Definitions

This appendix contains the complete OMG IDL [OMGIDL] for the Level 2 Document Object Model Core definitions.

The IDL files are also available as:

<http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/idl.zip>

dom.idl:

```
// File: dom.idl

#ifndef _DOM_IDL_
#define _DOM_IDL_

#pragma prefix "w3c.org"
module dom
{

    valuetype DOMString sequence<unsigned short>;

    typedef    unsigned long long DOMTimeStamp;

    interface DocumentType;
    interface Document;
    interface NodeList;
```

```
interface NamedNodeMap;
interface Element;

exception DOMException {
    unsigned short    code;
};
// ExceptionCode
const unsigned short    INDEX_SIZE_ERR                = 1;
const unsigned short    DOMSTRING_SIZE_ERR            = 2;
const unsigned short    HIERARCHY_REQUEST_ERR        = 3;
const unsigned short    WRONG_DOCUMENT_ERR           = 4;
const unsigned short    INVALID_CHARACTER_ERR        = 5;
const unsigned short    NO_DATA_ALLOWED_ERR          = 6;
const unsigned short    NO_MODIFICATION_ALLOWED_ERR   = 7;
const unsigned short    NOT_FOUND_ERR                 = 8;
const unsigned short    NOT_SUPPORTED_ERR             = 9;
const unsigned short    INUSE_ATTRIBUTE_ERR          = 10;
// Introduced in DOM Level 2:
const unsigned short    INVALID_STATE_ERR            = 11;
// Introduced in DOM Level 2:
const unsigned short    SYNTAX_ERR                   = 12;
// Introduced in DOM Level 2:
const unsigned short    INVALID_MODIFICATION_ERR     = 13;
// Introduced in DOM Level 2:
const unsigned short    NAMESPACE_ERR                = 14;
// Introduced in DOM Level 2:
const unsigned short    INVALID_ACCESS_ERR           = 15;

interface DOMImplementation {
    boolean                hasFeature(in DOMString feature,
                                     in DOMString version);

    // Introduced in DOM Level 2:
    DocumentType            createDocumentType(in DOMString qualifiedName,
                                              in DOMString publicId,
                                              in DOMString systemId)
                                raises(DOMException);

    // Introduced in DOM Level 2:
    Document                createDocument(in DOMString namespaceURI,
                                              in DOMString qualifiedName,
                                              in DocumentType doctype)
                                raises(DOMException);
};

interface Node {

    // NodeType
    const unsigned short    ELEMENT_NODE                = 1;
    const unsigned short    ATTRIBUTE_NODE              = 2;
    const unsigned short    TEXT_NODE                   = 3;
    const unsigned short    CDATA_SECTION_NODE          = 4;
    const unsigned short    ENTITY_REFERENCE_NODE       = 5;
    const unsigned short    ENTITY_NODE                 = 6;
    const unsigned short    PROCESSING_INSTRUCTION_NODE = 7;
    const unsigned short    COMMENT_NODE                = 8;
    const unsigned short    DOCUMENT_NODE               = 9;
    const unsigned short    DOCUMENT_TYPE_NODE          = 10;
    const unsigned short    DOCUMENT_FRAGMENT_NODE     = 11;
    const unsigned short    NOTATION_NODE               = 12;

    readonly attribute DOMString    nodeName;
    attribute DOMString             nodeValue;
```

```

// raises(DOMException) on setting
// raises(DOMException) on retrieval

readonly attribute unsigned short   nodeType;
readonly attribute Node              parentNode;
readonly attribute NodeList          childNodes;
readonly attribute Node              firstChild;
readonly attribute Node              lastChild;
readonly attribute Node              previousSibling;
readonly attribute Node              nextSibling;
readonly attribute NamedNodeMap      attributes;
// Modified in DOM Level 2:
readonly attribute Document          ownerDocument;
Node                                 insertBefore(in Node newChild,
                                                  in Node refChild)
                                                  raises(DOMException);
Node                                 replaceChild(in Node newChild,
                                                  in Node oldChild)
                                                  raises(DOMException);
Node                                 removeChild(in Node oldChild)
                                                  raises(DOMException);
Node                                 appendChild(in Node newChild)
                                                  raises(DOMException);

boolean                              hasChildNodes();
Node                                 cloneNode(in boolean deep);
// Modified in DOM Level 2:
void                                 normalize();
// Introduced in DOM Level 2:
boolean                              isSupported(in DOMString feature,
                                                  in DOMString version);

// Introduced in DOM Level 2:
readonly attribute DOMString          namespaceURI;
// Introduced in DOM Level 2:
    attribute DOMString                prefix;
// raises(DOMException) on setting

// Introduced in DOM Level 2:
readonly attribute DOMString          localName;
// Introduced in DOM Level 2:
boolean                              hasAttributes();
};

interface NodeList {
    Node                                item(in unsigned long index);
    readonly attribute unsigned long    length;
};

interface NamedNodeMap {
    Node                                getNamedItem(in DOMString name);
    Node                                setNamedItem(in Node arg)
                                          raises(DOMException);
    Node                                removeNamedItem(in DOMString name)
                                          raises(DOMException);
    Node                                item(in unsigned long index);
    readonly attribute unsigned long    length;
    // Introduced in DOM Level 2:
    Node                                getNamedItemNS(in DOMString namespaceURI,
                                                        in DOMString localName);
    // Introduced in DOM Level 2:
    Node                                setNamedItemNS(in Node arg)
                                                        raises(DOMException);
};
```



```
// Introduced in DOM Level 2:
Node                removeNamedItemNS(in DOMString namespaceURI,
                                       in DOMString localName)
                                       raises(DOMException);
};

interface CharacterData : Node {
    attribute DOMString    data;
                           // raises(DOMException) on setting
                           // raises(DOMException) on retrieval

    readonly attribute unsigned long    length;
    DOMString    substringData(in unsigned long offset,
                              in unsigned long count)
                              raises(DOMException);

    void    appendData(in DOMString arg)
            raises(DOMException);

    void    insertData(in unsigned long offset,
                      in DOMString arg)
            raises(DOMException);

    void    deleteData(in unsigned long offset,
                       in unsigned long count)
            raises(DOMException);

    void    replaceData(in unsigned long offset,
                       in unsigned long count,
                       in DOMString arg)
            raises(DOMException);
};

interface Attr : Node {
    readonly attribute DOMString    name;
    readonly attribute boolean    specified;
    attribute DOMString    value;
                           // raises(DOMException) on setting

    // Introduced in DOM Level 2:
    readonly attribute Element    ownerElement;
};

interface Element : Node {
    readonly attribute DOMString    tagName;
    DOMString    getAttribute(in DOMString name);
    void    setAttribute(in DOMString name,
                        in DOMString value)
            raises(DOMException);

    void    removeAttribute(in DOMString name)
            raises(DOMException);

    Attr    getAttributeNode(in DOMString name);
    Attr    setAttributeNode(in Attr newAttr)
            raises(DOMException);
    Attr    removeAttributeNode(in Attr oldAttr)
            raises(DOMException);

    NodeList    getElementsByTagName(in DOMString name);
    // Introduced in DOM Level 2:
    DOMString    getAttributeNS(in DOMString namespaceURI,
                               in DOMString localName);

    // Introduced in DOM Level 2:
    void    setAttributeNS(in DOMString namespaceURI,
                          in DOMString qualifiedName,
                          in DOMString value)
            raises(DOMException);
};
```

```
// Introduced in DOM Level 2:
void                removeAttributeNS(in DOMString namespaceURI,
                                      in DOMString localName)
                                      raises(DOMException);

// Introduced in DOM Level 2:
Attr                getAttributeNodeNS(in DOMString namespaceURI,
                                      in DOMString localName);

// Introduced in DOM Level 2:
Attr                setAttributeNodeNS(in Attr newAttr)
                                      raises(DOMException);

// Introduced in DOM Level 2:
NodeList            getElementsByTagNameNS(in DOMString namespaceURI,
                                      in DOMString localName);

// Introduced in DOM Level 2:
boolean             hasAttribute(in DOMString name);
// Introduced in DOM Level 2:
boolean             hasAttributeNS(in DOMString namespaceURI,
                                   in DOMString localName);
};

interface Text : CharacterData {
    Text            splitText(in unsigned long offset)
                    raises(DOMException);
};

interface Comment : CharacterData {
};

interface CDATASection : Text {
};

interface DocumentType : Node {
    readonly attribute DOMString        name;
    readonly attribute NamedNodeMap     entities;
    readonly attribute NamedNodeMap     notations;
    // Introduced in DOM Level 2:
    readonly attribute DOMString        publicId;
    // Introduced in DOM Level 2:
    readonly attribute DOMString        systemId;
    // Introduced in DOM Level 2:
    readonly attribute DOMString        internalSubset;
};

interface Notation : Node {
    readonly attribute DOMString        publicId;
    readonly attribute DOMString        systemId;
};

interface Entity : Node {
    readonly attribute DOMString        publicId;
    readonly attribute DOMString        systemId;
    readonly attribute DOMString        notationName;
};

interface EntityReference : Node {
};

interface ProcessingInstruction : Node {
    readonly attribute DOMString        target;
    attribute DOMString                data;
    // raises(DOMException) on setting
```

```
};

interface DocumentFragment : Node {
};

interface Document : Node {
    readonly attribute DocumentType      doctype;
    readonly attribute DOMImplementation implementation;
    readonly attribute Element           documentElement;
    Element                createElement(in DOMString tagName)
                                raises(DOMException);
    DocumentFragment       createDocumentFragment();
    Text                   createTextNode(in DOMString data);
    Comment                createComment(in DOMString data);
    CDATASection           createCDATASection(in DOMString data)
                                raises(DOMException);
    ProcessingInstruction   createProcessingInstruction(in DOMString target,
                                                        in DOMString data)
                                raises(DOMException);
    Attr                   createAttribute(in DOMString name)
                                raises(DOMException);
    EntityReference         createEntityReference(in DOMString name)
                                raises(DOMException);
    NodeList               getElementsByTagName(in DOMString tagname);
    // Introduced in DOM Level 2:
    Node                   importNode(in Node importedNode,
                                      in boolean deep)
                                raises(DOMException);
    // Introduced in DOM Level 2:
    Element                createElementNS(in DOMString namespaceURI,
                                          in DOMString qualifiedName)
                                raises(DOMException);
    // Introduced in DOM Level 2:
    Attr                  createAttributeNS(in DOMString namespaceURI,
                                           in DOMString qualifiedName)
                                raises(DOMException);
    // Introduced in DOM Level 2:
    NodeList              getElementsByTagNameNS(in DOMString namespaceURI,
                                                in DOMString localName);
    // Introduced in DOM Level 2:
    Element               getElementById(in DOMString elementId);
};
};

#endif // _DOM_IDL_
```

13 November, 2000

Appendix D: Java Language Binding

This appendix contains the complete Java Language [Java] binding for the Level 2 Document Object Model Core.

The Java files are also available as

<http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/java-binding.zip>

`org/w3c/dom/DOMException.java`:

```
package org.w3c.dom;
```

```
public class DOMException extends RuntimeException {
    public DOMException(short code, String message) {
        super(message);
        this.code = code;
    }
    public short    code;
    // ExceptionCode
    public static final short INDEX_SIZE_ERR           = 1;
    public static final short DOMSTRING_SIZE_ERR       = 2;
    public static final short HIERARCHY_REQUEST_ERR    = 3;
    public static final short WRONG_DOCUMENT_ERR       = 4;
    public static final short INVALID_CHARACTER_ERR    = 5;
    public static final short NO_DATA_ALLOWED_ERR      = 6;
    public static final short NO_MODIFICATION_ALLOWED_ERR = 7;
    public static final short NOT_FOUND_ERR            = 8;
    public static final short NOT_SUPPORTED_ERR        = 9;
    public static final short INUSE_ATTRIBUTE_ERR      = 10;
    public static final short INVALID_STATE_ERR        = 11;
    public static final short SYNTAX_ERR               = 12;
    public static final short INVALID_MODIFICATION_ERR = 13;
    public static final short NAMESPACE_ERR           = 14;
    public static final short INVALID_ACCESS_ERR       = 15;
}
}
```

org/w3c/dom/DOMImplementation.java:

```
package org.w3c.dom;

public interface DOMImplementation {
    public boolean hasFeature(String feature,
                              String version);

    public DocumentType createDocumentType(String qualifiedName,
                                           String publicId,
                                           String systemId)
        throws DOMException;

    public Document createDocument(String namespaceURI,
                                   String qualifiedName,
                                   DocumentType doctype)
        throws DOMException;
}
}
```

org/w3c/dom/DocumentFragment.java:

```
package org.w3c.dom;

public interface DocumentFragment extends Node {
}
```

org/w3c/dom/Document.java:

```
package org.w3c.dom;

public interface Document extends Node {
    public DocumentType getDoctype();

    public DOMImplementation getImplementation();
}
```

```
public Element getDocumentElement();

public Element createElement(String tagName)
    throws DOMException;

public DocumentFragment createDocumentFragment();

public Text createTextNode(String data);

public Comment createComment(String data);

public CDATASection createCDATASection(String data)
    throws DOMException;

public ProcessingInstruction createProcessingInstruction(String target,
    String data)
    throws DOMException;

public Attr createAttribute(String name)
    throws DOMException;

public EntityReference createEntityReference(String name)
    throws DOMException;

public NodeList getElementsByTagName(String tagname);

public Node importNode(Node importedNode,
    boolean deep)
    throws DOMException;

public Element createElementNS(String namespaceURI,
    String qualifiedName)
    throws DOMException;

public Attr createAttributeNS(String namespaceURI,
    String qualifiedName)
    throws DOMException;

public NodeList getElementsByTagNameNS(String namespaceURI,
    String localName);

public Element getElementById(String elementId);
}
```

org/w3c/dom/Node.java:

```
package org.w3c.dom;
```

```
public interface Node {
    // NodeType
    public static final short ELEMENT_NODE           = 1;
    public static final short ATTRIBUTE_NODE         = 2;
    public static final short TEXT_NODE              = 3;
    public static final short CDATA_SECTION_NODE     = 4;
    public static final short ENTITY_REFERENCE_NODE  = 5;
    public static final short ENTITY_NODE            = 6;
    public static final short PROCESSING_INSTRUCTION_NODE = 7;
    public static final short COMMENT_NODE           = 8;
    public static final short DOCUMENT_NODE          = 9;
    public static final short DOCUMENT_TYPE_NODE     = 10;
}
```

```
public static final short DOCUMENT_FRAGMENT_NODE    = 11;
public static final short NOTATION_NODE             = 12;

public String getNodeName();

public String getNodeValue()
    throws DOMException;
public void setNodeValue(String nodeValue)
    throws DOMException;

public short getNodeType();

public Node getParentNode();

public NodeList getChildNodes();

public Node getFirstChild();

public Node getLastChild();

public Node getPreviousSibling();

public Node getNextSibling();

public NamedNodeMap getAttributes();

public Document getOwnerDocument();

public Node insertBefore(Node newChild,
    Node refChild)
    throws DOMException;

public Node replaceChild(Node newChild,
    Node oldChild)
    throws DOMException;

public Node removeChild(Node oldChild)
    throws DOMException;

public Node appendChild(Node newChild)
    throws DOMException;

public boolean hasChildNodes();

public Node cloneNode(boolean deep);

public void normalize();

public boolean isSupported(String feature,
    String version);

public String getNamespaceURI();

public String getPrefix();
public void setPrefix(String prefix)
    throws DOMException;

public String getLocalName();

public boolean hasAttributes();
```

```
}
```

org/w3c/dom/NodeList.java:

```
package org.w3c.dom;
```

```
public interface NodeList {  
    public Node item(int index);  
  
    public int getLength();  
}
```

org/w3c/dom/NamedNodeMap.java:

```
package org.w3c.dom;
```

```
public interface NamedNodeMap {  
    public Node getNamedItem(String name);  
  
    public Node setNamedItem(Node arg)  
        throws DOMException;  
  
    public Node removeNamedItem(String name)  
        throws DOMException;  
  
    public Node item(int index);  
  
    public int getLength();  
  
    public Node getNamedItemNS(String namespaceURI,  
                               String localName);  
  
    public Node setNamedItemNS(Node arg)  
        throws DOMException;  
  
    public Node removeNamedItemNS(String namespaceURI,  
                                   String localName)  
        throws DOMException;  
}
```

org/w3c/dom/CharacterData.java:

```
package org.w3c.dom;
```

```
public interface CharacterData extends Node {  
    public String getData()  
        throws DOMException;  
    public void setData(String data)  
        throws DOMException;  
  
    public int getLength();  
  
    public String substringData(int offset,  
                                int count)  
        throws DOMException;  
  
    public void appendData(String arg)  
        throws DOMException;
```

```
    public void insertData(int offset,
                           String arg)
                           throws DOMException;

    public void deleteData(int offset,
                           int count)
                           throws DOMException;

    public void replaceData(int offset,
                            int count,
                            String arg)
                            throws DOMException;
}
```

org/w3c/dom/Attr.java:

```
package org.w3c.dom;
```

```
public interface Attr extends Node {
    public String getName();

    public boolean getSpecified();

    public String getValue();
    public void setValue(String value)
                      throws DOMException;

    public Element getOwnerElement();
}
```

org/w3c/dom/Element.java:

```
package org.w3c.dom;
```

```
public interface Element extends Node {
    public String getTagName();

    public String getAttribute(String name);

    public void setAttribute(String name,
                             String value)
                             throws DOMException;

    public void removeAttribute(String name)
                             throws DOMException;

    public Attr getAttributeNode(String name);

    public Attr setAttributeNode(Attr newAttr)
                             throws DOMException;

    public Attr removeAttributeNode(Attr oldAttr)
                             throws DOMException;

    public NodeList getElementsByTagName(String name);

    public String getAttributeNS(String namespaceURI,
                                  String localName);
}
```



```
public void setAttributeNS(String namespaceURI,
                           String qualifiedName,
                           String value)
    throws DOMException;

public void removeAttributeNS(String namespaceURI,
                              String localName)
    throws DOMException;

public Attr getAttributeNodeNS(String namespaceURI,
                               String localName);

public Attr setAttributeNodeNS(Attr newAttr)
    throws DOMException;

public NodeList getElementsByTagNameNS(String namespaceURI,
                                       String localName);

public boolean hasAttribute(String name);

public boolean hasAttributeNS(String namespaceURI,
                              String localName);
}
```

org/w3c/dom/Text.java:

```
package org.w3c.dom;
```

```
public interface Text extends CharacterData {
    public Text splitText(int offset)
        throws DOMException;
}
```

org/w3c/dom/Comment.java:

```
package org.w3c.dom;
```

```
public interface Comment extends CharacterData {
}
```

org/w3c/dom/CDATASection.java:

```
package org.w3c.dom;
```

```
public interface CDATASection extends Text {
}
```

org/w3c/dom/DocumentType.java:

```
package org.w3c.dom;
```

```
public interface DocumentType extends Node {
    public String getName();

    public NamedNodeMap getEntities();

    public NamedNodeMap getNotations();

    public String getPublicId();
}
```

```
    public String getSystemId();

    public String getInternalSubset();
}

org/w3c/dom/Notation.java:

package org.w3c.dom;

public interface Notation extends Node {
    public String getPublicId();

    public String getSystemId();
}

org/w3c/dom/Entity.java:

package org.w3c.dom;

public interface Entity extends Node {
    public String getPublicId();

    public String getSystemId();

    public String getNotationName();
}

org/w3c/dom/EntityReference.java:

package org.w3c.dom;

public interface EntityReference extends Node {
}

org/w3c/dom/ProcessingInstruction.java:

package org.w3c.dom;

public interface ProcessingInstruction extends Node {
    public String getTarget();

    public String getData();
    public void setData(String data)
        throws DOMException;
}
```

13 November, 2000

Appendix E: ECMAScript Language Binding

This appendix contains the complete ECMAScript [ECMAScript] binding for the Level 2 Document Object Model Core definitions.

Note: Exceptions handling is only supported by ECMAScript implementation conformant with the Standard ECMA-262 3rd. Edition ([ECMAScript]).

Prototype Object DOMException

The DOMException class has the following constants:

DOMException.INDEX_SIZE_ERR

This constant is of type Number and its value is 1.

DOMException.DOMSTRING_SIZE_ERR

This constant is of type Number and its value is 2.

DOMException.HIERARCHY_REQUEST_ERR

This constant is of type Number and its value is 3.

DOMException.WRONG_DOCUMENT_ERR

This constant is of type Number and its value is 4.

DOMException.INVALID_CHARACTER_ERR

This constant is of type Number and its value is 5.

DOMException.NO_DATA_ALLOWED_ERR

This constant is of type Number and its value is 6.

DOMException.NO_MODIFICATION_ALLOWED_ERR

This constant is of type Number and its value is 7.

DOMException.NOT_FOUND_ERR

This constant is of type Number and its value is 8.

DOMException.NOT_SUPPORTED_ERR

This constant is of type Number and its value is 9.

DOMException.INUSE_ATTRIBUTE_ERR

This constant is of type Number and its value is 10.

DOMException.INVALID_STATE_ERR

This constant is of type Number and its value is 11.

DOMException.SYNTAX_ERR

This constant is of type Number and its value is 12.

DOMException.INVALID_MODIFICATION_ERR

This constant is of type Number and its value is 13.

DOMException.NAMESPACE_ERR

This constant is of type Number and its value is 14.

DOMException.INVALID_ACCESS_ERR

This constant is of type Number and its value is 15.

Object DOMException

The DOMException object has the following properties:

code

This property is of type Number.

Object DOMImplementation

The DOMImplementation object has the following methods:

hasFeature(feature, version)

This method returns a Boolean.

The feature parameter is of type String.

The version parameter is of type String.

createDocumentType(qualifiedName, publicId, systemId)

This method returns a DocumentType object.

The qualifiedName parameter is of type String.

The publicId parameter is of type String.

The systemId parameter is of type String.

This method can raise a DOMException object.

createDocument(namespaceURI, qualifiedName, doctype)

This method returns a Document object.

The namespaceURI parameter is of type String.

The qualifiedName parameter is of type String.

The doctype parameter is a DocumentType object.

This method can raise a DOMException object.

Object DocumentFragment

DocumentFragment has the all the properties and methods of the Node object as well as the properties and methods defined below.

Object Document

Document has the all the properties and methods of the Node object as well as the properties and methods defined below.

The Document object has the following properties:

doctype

This read-only property is a `DocumentType` object.

implementation

This read-only property is a `DOMImplementation` object.

documentElement

This read-only property is a `Element` object.

The `Document` object has the following methods:

`createElement(tagName)`

This method returns a `Element` object.

The `tagName` parameter is of type `String`.

This method can raise a `DOMException` object.

`createDocumentFragment()`

This method returns a `DocumentFragment` object.

`createTextNode(data)`

This method returns a `Text` object.

The `data` parameter is of type `String`.

`createComment(data)`

This method returns a `Comment` object.

The `data` parameter is of type `String`.

`createCDATASection(data)`

This method returns a `CDATASection` object.

The `data` parameter is of type `String`.

This method can raise a `DOMException` object.

`createProcessingInstruction(target, data)`

This method returns a `ProcessingInstruction` object.

The `target` parameter is of type `String`.

The `data` parameter is of type `String`.

This method can raise a `DOMException` object.

`createAttribute(name)`

This method returns a `Attr` object.

The `name` parameter is of type `String`.

This method can raise a `DOMException` object.

`createEntityReference(name)`

This method returns a `EntityReference` object.

The `name` parameter is of type `String`.

This method can raise a `DOMException` object.

`getElementsByTagName(tagname)`

This method returns a `NodeList` object.

The `tagname` parameter is of type `String`.

`importNode(importedNode, deep)`

This method returns a `Node` object.

The `importedNode` parameter is a `Node` object.

The `deep` parameter is of type `Boolean`.

This method can raise a `DOMException` object.

`createElementNS(namespaceURI, qualifiedName)`

This method returns a `Element` object.

The `namespaceURI` parameter is of type `String`.

The `qualifiedName` parameter is of type `String`.

This method can raise a `DOMException` object.

`createAttributeNS(namespaceURI, qualifiedName)`

This method returns a `Attr` object.

The `namespaceURI` parameter is of type `String`.

The `qualifiedName` parameter is of type `String`.

This method can raise a `DOMException` object.

`getElementsByTagNameNS(namespaceURI, localName)`

This method returns a `NodeList` object.

The `namespaceURI` parameter is of type `String`.

The `localName` parameter is of type `String`.

`getElementById(elementId)`

This method returns a `Element` object.

The `elementId` parameter is of type `String`.

Prototype Object Node

The Node class has the following constants:

Node.ELEMENT_NODE
This constant is of type Number and its value is 1.

Node.ATTRIBUTE_NODE
This constant is of type Number and its value is 2.

Node.TEXT_NODE
This constant is of type Number and its value is 3.

Node.CDATA_SECTION_NODE
This constant is of type Number and its value is 4.

Node.ENTITY_REFERENCE_NODE
This constant is of type Number and its value is 5.

Node.ENTITY_NODE
This constant is of type Number and its value is 6.

Node.PROCESSING_INSTRUCTION_NODE
This constant is of type Number and its value is 7.

Node.COMMENT_NODE
This constant is of type Number and its value is 8.

Node.DOCUMENT_NODE
This constant is of type Number and its value is 9.

Node.DOCUMENT_TYPE_NODE
This constant is of type Number and its value is 10.

Node.DOCUMENT_FRAGMENT_NODE
This constant is of type Number and its value is 11.

Node.NOTATION_NODE
This constant is of type Number and its value is 12.

Object Node

The Node object has the following properties:

nodeName
This read-only property is of type String.

nodeValue
This property is of type String, can raise a DOMException object on setting and can raise a DOMException object on retrieval.

nodeType
This read-only property is of type Number.

parentNode
This read-only property is a Node object.

childNodes
This read-only property is a NodeList object.

firstChild
This read-only property is a Node object.

lastChild
This read-only property is a Node object.

previousSibling
This read-only property is a Node object.

nextSibling
This read-only property is a Node object.

attributes
This read-only property is a NamedNodeMap object.

ownerDocument
This read-only property is a Document object.

namespaceURI
This read-only property is of type String.

prefix
This property is of type String and can raise a DOMException object on setting.

localName
This read-only property is of type String.

The Node object has the following methods:

insertBefore(newChild, refChild)

This method returns a Node object.
The newChild parameter is a Node object.
The refChild parameter is a Node object.
This method can raise a DOMException object.
replaceChild(newChild, oldChild)
This method returns a Node object.
The newChild parameter is a Node object.
The oldChild parameter is a Node object.
This method can raise a DOMException object.
removeChild(oldChild)
This method returns a Node object.
The oldChild parameter is a Node object.
This method can raise a DOMException object.
appendChild(newChild)
This method returns a Node object.
The newChild parameter is a Node object.
This method can raise a DOMException object.
hasChildNodes()
This method returns a Boolean.
cloneNode(deep)
This method returns a Node object.
The deep parameter is of type Boolean.
normalize()
This method has no return value.
isSupported(feature, version)
This method returns a Boolean.
The feature parameter is of type String.
The version parameter is of type String.
hasAttributes()
This method returns a Boolean.

Object NodeList

The NodeList object has the following properties:

length

This read-only property is of type Number.

The NodeList object has the following methods:

item(index)

This method returns a Node object.

The index parameter is of type Number.

Note: This object can also be dereferenced using square bracket notation (e.g. obj[1]). Dereferencing with an integer index is equivalent to invoking the item method with that index.

Object NamedNodeMap

The NamedNodeMap object has the following properties:

length

This read-only property is of type Number.

The NamedNodeMap object has the following methods:

getNamedItem(name)

This method returns a Node object.

The name parameter is of type String.

setNamedItem(arg)

This method returns a Node object.

The arg parameter is a Node object.

This method can raise a DOMException object.

removeNamedItem(name)

This method returns a Node object.

The name parameter is of type String.

This method can raise a DOMException object.

item(index)

This method returns a Node object.

The index parameter is of type Number.

Note: This object can also be dereferenced using square bracket notation (e.g. `obj[1]`). Dereferencing with an integer index is equivalent to invoking the `item` method with that index.

`getNamedItemNS(namespaceURI, localName)`

This method returns a Node object.

The `namespaceURI` parameter is of type String.

The `localName` parameter is of type String.

`setNamedItemNS(arg)`

This method returns a Node object.

The `arg` parameter is a Node object.

This method can raise a DOMException object.

`removeNamedItemNS(namespaceURI, localName)`

This method returns a Node object.

The `namespaceURI` parameter is of type String.

The `localName` parameter is of type String.

This method can raise a DOMException object.

Object `CharacterData`

`CharacterData` has all the properties and methods of the Node object as well as the properties and methods defined below.

The `CharacterData` object has the following properties:

`data`

This property is of type String, can raise a DOMException object on setting and can raise a DOMException object on retrieval.

`length`

This read-only property is of type Number.

The `CharacterData` object has the following methods:

`substringData(offset, count)`

This method returns a String.

The `offset` parameter is of type Number.

The `count` parameter is of type Number.

This method can raise a DOMException object.

`appendData(arg)`

This method has no return value.

The `arg` parameter is of type String.

This method can raise a DOMException object.

`insertData(offset, arg)`

This method has no return value.

The `offset` parameter is of type Number.

The `arg` parameter is of type String.

This method can raise a DOMException object.

`deleteData(offset, count)`

This method has no return value.

The `offset` parameter is of type Number.

The `count` parameter is of type Number.

This method can raise a DOMException object.

`replaceData(offset, count, arg)`

This method has no return value.

The `offset` parameter is of type Number.

The `count` parameter is of type Number.

The `arg` parameter is of type String.

This method can raise a DOMException object.

Object `Attr`

`Attr` has all the properties and methods of the Node object as well as the properties and methods defined below.

The `Attr` object has the following properties:

`name`

This read-only property is of type String.

`specified`

This read-only property is of type Boolean.

value

This property is of type String and can raise a DOMException object on setting.

ownerElement

This read-only property is a Element object.

Object Element

Element has the all the properties and methods of the Node object as well as the properties and methods defined below.

The Element object has the following properties:

tagName

This read-only property is of type String.

The Element object has the following methods:

getAttribute(name)

This method returns a String.

The name parameter is of type String.

setAttribute(name, value)

This method has no return value.

The name parameter is of type String.

The value parameter is of type String.

This method can raise a DOMException object.

removeAttribute(name)

This method has no return value.

The name parameter is of type String.

This method can raise a DOMException object.

getAttributeNode(name)

This method returns a Attr object.

The name parameter is of type String.

setAttributeNode(newAttr)

This method returns a Attr object.

The newAttr parameter is a Attr object.

This method can raise a DOMException object.

removeAttributeNode(oldAttr)

This method returns a Attr object.

The oldAttr parameter is a Attr object.

This method can raise a DOMException object.

getElementsByTagName(name)

This method returns a NodeList object.

The name parameter is of type String.

getAttributeNS(namespaceURI, localName)

This method returns a String.

The namespaceURI parameter is of type String.

The localName parameter is of type String.

setAttributeNS(namespaceURI, qualifiedName, value)

This method has no return value.

The namespaceURI parameter is of type String.

The qualifiedName parameter is of type String.

The value parameter is of type String.

This method can raise a DOMException object.

removeAttributeNS(namespaceURI, localName)

This method has no return value.

The namespaceURI parameter is of type String.

The localName parameter is of type String.

This method can raise a DOMException object.

getAttributeNodeNS(namespaceURI, localName)

This method returns a Attr object.

The namespaceURI parameter is of type String.

The localName parameter is of type String.

setAttributeNodeNS(newAttr)

This method returns a Attr object.

The newAttr parameter is a Attr object.

This method can raise a DOMException object.

`getElementsByTagNameNS(namespaceURI, localName)`
This method returns a `NodeList` object.
The `namespaceURI` parameter is of type `String`.
The `localName` parameter is of type `String`.
`hasAttribute(name)`
This method returns a `Boolean`.
The `name` parameter is of type `String`.
`hasAttributeNS(namespaceURI, localName)`
This method returns a `Boolean`.
The `namespaceURI` parameter is of type `String`.
The `localName` parameter is of type `String`.

Object Text

Text has the all the properties and methods of the `CharacterData` object as well as the properties and methods defined below.

The Text object has the following methods:

`splitText(offset)`
This method returns a Text object.
The `offset` parameter is of type `Number`.
This method can raise a `DOMException` object.

Object Comment

Comment has the all the properties and methods of the `CharacterData` object as well as the properties and methods defined below.

Object CDATASection

CDATASection has the all the properties and methods of the Text object as well as the properties and methods defined below.

Object DocumentType

DocumentType has the all the properties and methods of the Node object as well as the properties and methods defined below.

The DocumentType object has the following properties:

`name`
This read-only property is of type `String`.
`entities`
This read-only property is a `NamedNodeMap` object.
`notations`
This read-only property is a `NamedNodeMap` object.
`publicId`
This read-only property is of type `String`.
`systemId`
This read-only property is of type `String`.
`internalSubset`
This read-only property is of type `String`.

Object Notation

Notation has the all the properties and methods of the Node object as well as the properties and methods defined below.

The Notation object has the following properties:

`publicId`
This read-only property is of type `String`.
`systemId`
This read-only property is of type `String`.

Object Entity

Entity has the all the properties and methods of the Node object as well as the properties and methods defined below.

The Entity object has the following properties:

`publicId`
This read-only property is of type `String`.
`systemId`
This read-only property is of type `String`.
`notationName`
This read-only property is of type `String`.

Object EntityReference

EntityReference has the all the properties and methods of the Node

object as well as the properties and methods defined below.

Object ProcessingInstruction

ProcessingInstruction has the all the properties and methods of the Node object as well as the properties and methods defined below.

The ProcessingInstruction object has the following properties:

target

This read-only property is of type String.

data

This property is of type String and can raise a DOMException object on setting.

13 November, 2000

Appendix F: Acknowledgements

Many people contributed to this specification, including members of the DOM Working Group and the DOM Interest Group. We especially thank the following:

Lauren Wood (SoftQuad Software Inc., chair), Andrew Watson (Object Management Group), Andy Heninger (IBM), Arnaud Le Hors (W3C and IBM), Ben Chang (Oracle), Bill Smith (Sun), Bill Shea (Merrill Lynch), Bob Sutor (IBM), Chris Lovett (Microsoft), Chris Wilson (Microsoft), David Brownell (Sun), David Singer (IBM), Don Park (invited), Eric Vasilik (Microsoft), Gavin Nicol (INSO), Ian Jacobs (W3C), James Clark (invited), James Davidson (Sun), Jared Sorensen (Novell), Joe Kesselman (IBM), Joe Lapp (webMethods), Joe Marini (Macromedia), Johnny Stenback (Netscape), Jonathan Marsh (Microsoft), Jonathan Robie (Texcel Research and Software AG), Kim Adamson-Sharpe (SoftQuad Software Inc.), Laurence Cable (Sun), Mark Davis (IBM), Mark Scardina (Oracle), Martin Dürst (W3C), Mick Goulish (Software AG), Mike Champion (Arbortext and Software AG), Miles Sabin (Cromwell Media), Patti Lutsky (Arbortext), Paul Grosso (Arbortext), Peter Sharpe (SoftQuad Software Inc.), Phil Karlton (Netscape), Philippe Le Hégarret (W3C, W3C team contact), Ramesh Lekshmyanarayanan (Merrill Lynch), Ray Whitmer (iMall, Excite@Home and Netscape), Rich Rollman (Microsoft), Rick Gessner (Netscape), Scott Isaacs (Microsoft), Sharon Adler (INSO), Steve Byrne (JavaSoft), Tim Bray (invited), Tom Pixley (Netscape), Vidur Apparao (Netscape), Vinod Anupam (Lucent).

Thanks to all those who have helped to improve this specification by sending suggestions and corrections.

F.1: Production Systems

This specification was written in XML. The HTML, OMG IDL, Java and ECMA Script bindings were all produced automatically.

Thanks to Joe English, author of cost, which was used as the basis for producing DOM Level 1. Thanks also to Gavin Nicol, who wrote the scripts which run on top of cost. Arnaud Le Hors and Philippe Le Hégarret maintained the scripts.

For DOM Level 2, we used Xerces as the basis DOM implementation and wish to thank the authors. Philippe Le Hégarret and Arnaud Le Hors wrote the Java programs which are the DOM application.

Thanks also to Jan Kärroman, author of html2ps, which we use in creating the PostScript version of the specification.

13 November, 2000

Glossary

Editors

Arnaud Le Hors, IBM
Lauren Wood, SoftQuad Software Inc.
Robert S. Sutor, IBM (for DOM Level 1)

Several of the following term definitions have been borrowed or modified from similar definitions in other W3C or standards documents. See the links within the definitions for more information.

16-bit unit

The base unit of a DOMString. This indicates that indexing on a DOMString occurs in units of 16 bits. This must not be misunderstood to mean that a DOMString can store arbitrary 16-bit units. A DOMString is a character string encoded in UTF-16; this means that the restrictions of UTF-16 as well as the other relevant restrictions on character strings must be maintained. A single character, for example in the form of a numeric character reference, may correspond to one or two 16-bit units.

For more information, see [Unicode] and [ISO/IEC 10646].

ancestor

An ancestor node of any node A is any node above A in a tree model of a document, where "above" means "toward the root."

API

An API is an application programming interface, a set of functions or methods used to access some functionality.

child

A child is an immediate descendant node of a node.

client application

A [client] application is any software that uses the Document Object Model programming interfaces provided by the hosting implementation to accomplish useful work. Some examples of client applications are scripts within an HTML or XML document.

COM

COM is Microsoft's Component Object Model [COM], a technology for building applications from binary software components.

convenience

A convenience method is an operation on an object that could be accomplished by a program consisting of more basic operations on the object. Convenience methods are usually provided to make the API easier and simpler to use or to allow specific programs to create more optimized implementations for common operations. A similar definition holds for a convenience property.

data model

A data model is a collection of descriptions of data structures and their contained fields, together with the operations or functions that manipulate them.

descendant

A descendant node of any node A is any node below A in a tree model of a document, where "above" means "toward the root."

ECMAScript

The programming language defined by the ECMA-262 standard [ECMAScript]. As stated in the standard, the originating technology for ECMAScript was JavaScript [JavaScript]. Note that in the ECMAScript Language binding, the word "property" is used in the same sense as the IDL term "attribute."

element

Each document contains one or more elements, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements by an empty-element tag. Each element has a type, identified by name, and may have a set of attributes. Each attribute has a name and a

value. See Logical Structures in XML [XML].

information item

An information item is an abstract representation of some component of an XML document. See the [InfoSet] for details.

hosting implementation

A [hosting] implementation is a software module that provides an implementation of the DOM interfaces so that a client application can use them. Some examples of hosting implementations are browsers, editors and document repositories.

HTML

The HyperText Markup Language (HTML) is a simple markup language used to create hypertext documents that are portable from one platform to another. HTML documents are SGML documents with generic semantics that are appropriate for representing information from a wide range of applications. [HTML4.0]

inheritance

In object-oriented programming, the ability to create new classes (or interfaces) that contain all the methods and properties of another class (or interface), plus additional methods and properties. If class (or interface) D inherits from class (or interface) B, then D is said to be derived from B. B is said to be a base class (or interface) for D. Some programming languages allow for multiple inheritance, that is, inheritance from more than one class or interface.

interface

An interface is a declaration of a set of methods with no information given about their implementation. In object systems that support interfaces and inheritance, interfaces can usually inherit from one another.

language binding

A programming language binding for an IDL specification is an implementation of the interfaces in the specification for the given language. For example, a Java language binding for the Document Object Model IDL specification would implement the concrete Java classes that provide the functionality exposed by the interfaces.

local name

A local name is the local part of a qualified name. This is called the local part in Namespaces in XML [Namespaces].

method

A method is an operation or function that is associated with an object and is allowed to manipulate the object's data.

model

A model is the actual data representation for the information at hand. Examples are the structural model and the style model representing the parse structure and the style information associated with a document. The model might be a tree, or a directed graph, or something else.

namespace prefix

A namespace prefix is a string that associates an element or attribute name with a namespace URI in XML. See namespace prefix in Namespaces in XML [Namespaces].

namespace URI

A namespace URI is a URI that identifies an XML namespace. Strictly speaking, this actually is a namespace URI reference. This is called the namespace name in Namespaces in XML [Namespaces].

object model

An object model is a collection of descriptions of classes or interfaces, together with their member data, member functions, and class-static operations.

parent

A parent is an immediate ancestor node of a node.

qualified name

A qualified name is the name of an element or attribute defined as the

concatenation of a local name (as defined in this specification), optionally preceded by a namespace prefix and colon character. See Qualified Names in Namespaces in XML [Namespaces].

readonly node

A readonly node is a node that is immutable. This means its list of children, its content, and its attributes, when it is an element, cannot be changed in any way. However, a readonly node can possibly be moved, when it is not itself contained in a readonly node.

root node

The root node is the unique node that is not a child of any other node. All other nodes are children or other descendants of the root node.

sibling

Two nodes are siblings if and only if they have the same parent node.

string comparison

When string matching is required, it is to occur as though the comparison was between 2 sequences of code points from the Unicode 3.0 standard [Unicode].

token

An information item such as an XML Name which has been tokenized.

tokenized

The description given to various information items (for example, attribute values of various types, but not including the StringType CDATA) after having been processed by the XML processor. The process includes stripping leading and trailing white space, and replacing multiple space characters by one. See the definition of tokenized type.

well-formed document

A document is well-formed if it is tag valid and entities are limited to single elements (i.e., single sub-trees). See Well-Formed XML Documents in XML [XML].

XML

Extensible Markup Language (XML) is an extremely simple dialect of SGML. The goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML [XML] has been designed for ease of implementation and for interoperability with both SGML and HTML.

XML name

See XML name in the XML specification [XML].

XML namespace

An XML namespace is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute names. [Namespaces]

13 November, 2000

References

For the latest version of any W3C specification please consult the list of W3C Technical Reports available at <http://www.w3.org/TR>.

H.1: Normative references

Charmod

W3C (World Wide Web Consortium) Character Model for the World Wide Web, November 1999. Available at <http://www.w3.org/TR/1999/WD-charmod-19991129>

ECMAScript

ECMA (European Computer Manufacturers Association) ECMAScript Language Specification. Available at <http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM>

HTML4.0

W3C (World Wide Web Consortium) HTML 4.0 Specification, April 1998. Available at <http://www.w3.org/TR/1998/REC-html40-19980424>

ISO/IEC 10646

ISO (International Organization for Standardization). ISO/IEC 10646-1:2000 (E). Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane. [Geneva]: International Organization for Standardization.

Java

Sun Microsystems Inc. The Java Language Specification, James Gosling, Bill Joy, and Guy Steele, September 1996. Available at <http://java.sun.com/docs/books/jls>

Namespaces

W3C (World Wide Web Consortium) Namespaces in XML, January 1999. Available at <http://www.w3.org/TR/1999/REC-xml-names-19990114>

OMGIDL

OMG (Object Management Group) IDL (Interface Definition Language) defined in The Common Object Request Broker: Architecture and Specification, version 2.3.1, October 1999. Available from <http://www.omg.org/>

RFC2396

IETF (Internet Engineering Task Force) RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, eds. T. Berners-Lee, R. Fielding, L. Masinter. August 1998. Available at <http://www.ietf.org/rfc/rfc2396.txt>

Unicode

The Unicode Consortium. The Unicode Standard, Version 3.0., February 2000. Available at <http://www.unicode.org/unicode/standard/versions/Unicode3.0.html>.

XML

W3C (World Wide Web Consortium) Extensible Markup Language (XML) 1.0, February 1998. Available at <http://www.w3.org/TR/1998/REC-xml-19980210>

H.2: Informative references

DOM Level 2 CSS

W3C (World Wide Web Consortium) Document Object Model Level 2 CSS. Available at <http://www.w3.org/TR/DOM-Level-2-Style/css>

COM

Microsoft Corp. The Component Object Model. Available at <http://www.microsoft.com/com>

CORBA

OMG (Object Management Group) The Common Object Request Broker: Architecture and Specification, version 2.3.1, October 1999. Available from <http://www.omg.org/>

DOM Level 1

W3C (World Wide Web Consortium) DOM Level 1 Specification, October 1998. Available at <http://www.w3.org/TR/REC-DOM-Level-1>

DOM Level 2 HTML

W3C (World Wide Web Consortium) Document Object Model Level 2 HTML Specification. Available at <http://www.w3.org/TR/DOM-Level-2-HTML>

DOM Level 2 Events

W3C (World Wide Web Consortium) Document Object Model Level 2 Events Specification. Available at <http://www.w3.org/TR/DOM-Level-2-Events>

InfoSet

W3C (World Wide Web Consortium) XML Information Set, December 1999. Available at <http://www.w3.org/TR/xml-infoSet>

JavaIDL

Sun Microsystems Inc. Java IDL. Available at <http://java.sun.com/products/jdk/1.2/docs/guide/idl>

JavaScript

Netscape Communications Corp. JavaScript Resources. Available at <http://developer.netscape.com/tech/javascript/resources.html>

JScript

Microsoft Corp. JScript Resources. Available at

<http://msdn.microsoft.com/scripting/default.htm>

MIDL

Microsoft Corp. MIDL Language Reference. Available at
http://msdn.microsoft.com/library/psdk/midl/mi-laref_1rlh.htm

DOM Level 2 Style Sheets

W3C (World Wide Web Consortium) Document Object Model Level 2 Style Sheets. Available at <http://www.w3.org/TR/DOM-Level-2-Style/stylesheets>

DOM Level 2 Traversal

W3C (World Wide Web Consortium) Document Object Model Level 2 Traversal. Available at
<http://www.w3.org/TR/DOM-Level-2-Traversal-Range/traversal>

DOM Level 2 Range

W3C (World Wide Web Consortium) Document Object Model Level 2 Range. Available at <http://www.w3.org/TR/DOM-Level-2-Traversal-Range/ranges>

DOM Level 2 Views

W3C (World Wide Web Consortium) Document Object Model Level 2 Views Specification. Available at <http://www.w3.org/TR/DOM-Level-2-Views>

XPointer

W3C (World Wide Web Consortium) XML Pointer Language (XPointer), June 2000. Available at <http://www.w3.org/TR/xptr>

13 November, 2000

Index

16-bit unit 1, 2, 3, 4, 5,
6, 7, 8, 9

ancestor 1, 2, 3, 4
appendData
attributes

API 1, 2, 3, 4, 5, 6
Attr

appendChild
ATTRIBUTE_NODE

CDATA_SECTION_NODE
Charmod 1, 2
client application 1, 2
Comment
CORBA 1, 2
createCDATASection
createDocumentFragment
createElementNS
createTextNode

CDATASection
child 1, 2, 3
cloneNode
COMMENT_NODE
createAttribute
createComment
createDocumentType
createEntityReference

CharacterData
childNodes
COM 1, 2, 3, 4
convenience 1, 2, 3
createAttributeNS
createDocument
createElement
createProcessingInstruction

data 1, 2
descendant 1, 2, 3, 4, 5,
6, 7
DOCUMENT_FRAGMENT_NODE
documentElement
DOM Level 1 1, 2

data model 1, 2
doctype
DOCUMENT_NODE
DocumentFragment
DOM Level 2 CSS 1, 2

deleteData

Document
DOCUMENT_TYPE_NODE
DocumentType
DOM Level 2 Events 1, 2

DOM Level 2 HTML 1, 2, 3, 4
DOM Level 2 Range 1, 2
DOM Level 2 Traversal 1, 2
DOMImplementation
DOMTimeStamp
DOM Level 2 Views 1, 2
DOMString

DOM Level 2 Style Sheets 1,
2
DOMException
DOMSTRING_SIZE_ERR

ECMAScript 1, 2, 3, 4
entities
ENTITY_REFERENCE_NODE

Element 1, 2, 3, 4, 5, 6
Entity
EntityReference

ELEMENT_NODE
ENTITY_NODE

firstChild

getAttribute	getAttributeNode	getAttributeNodeNS
getAttributeNS	getElementById	getElementsByTagName 1, 2
getElementsByTagNameNS 1, 2	getNamedItem	getNamedItemNS
hasAttribute	hasAttributeNS	hasAttributes
hasChildNodes	hasFeature	HIERARCHY_REQUEST_ERR
hosting implementation 1, 2	HTML 1, 2	HTML4.0 1, 2
implementation	importNode	INDEX_SIZE_ERR
information item 1, 2	InfoSet 1, 2, 3, 4	inheritance 1, 2
insertBefore	insertData	interface 1, 2
internalSubset	INUSE_ATTRIBUTE_ERR	INVALID_ACCESS_ERR
INVALID_CHARACTER_ERR	INVALID_MODIFICATION_ERR	INVALID_STATE_ERR
ISO/IEC 10646 1, 2, 3	isSupported	item 1, 2
Java 1, 2	JavaIDL 1, 2	JavaScript 1, 2, 3
JScript 1, 2		
language binding 1, 2	lastChild	length 1, 2, 3
live 1, 2, 3	local name 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	localName
method 1, 2	MIDL 1, 2	model 1, 2
name 1, 2	NamedNodeMap	namespace prefix 1, 2, 3, 4, 5, 6
namespace URI 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17	NAMESPACE_ERR	Namespaces 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
namespaceURI	nextSibling	NO_DATA_ALLOWED_ERR
NO_MODIFICATION_ALLOWED_ERR	Node	NodeList
nodeName	nodeType	nodeValue
normalize	NOT_FOUND_ERR	NOT_SUPPORTED_ERR
Notation	NOTATION_NODE	notationName
notations		
object model 1, 2, 3	OMGIDL 1, 2, 3	ownerDocument
ownerElement		
parent 1, 2	parentNode	prefix
previousSibling	PROCESSING_INSTRUCTION_NODE	ProcessingInstruction
publicId 1, 2, 3		
qualified name 1, 2, 3, 4, 5, 6, 7, 8, 9		
readonly node 1, 2, 3, 4, 5	removeAttribute	removeAttributeNode
removeAttributeNS	removeChild	removeNamedItem
removeNamedItemNS	replaceChild	replaceData
RFC2396 1, 2	root node 1, 2	
setAttribute	setAttributeNode	setAttributeNodeNS
setAttributeNS	setNamedItem	setNamedItemNS
sibling 1, 2, 3	specified	splitText
string comparison 1, 2, 3	substringData	SYNTAX_ERR
systemId 1, 2, 3		
tagName	target	Text
TEXT_NODE	token 1, 2	tokenized 1, 2

Unicode 1, 2, 3, 4

value

well-formed document 1, 2 WRONG_DOCUMENT_ERR

XML 1, 2, 3, 4, 5, 6, 7 XML name 1, 2

XML namespace 1, 2

XPointer 1, 2



Document Object Model (DOM) Level 2 Core Specification

Version 1.0

W3C Recommendation 13 November, 2000

This version:

<http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113>
(PostScript file , PDF file , plain text , ZIP file)

Latest version:

<http://www.w3.org/TR/DOM-Level-2-Core>

Previous version:

<http://www.w3.org/TR/2000/PR-DOM-Level-2-Core-20000927>

Editors:

Arnaud Le Hors, *W3C team contact until October 1999, then IBM*

Philippe Le Hégarret, *W3C, team contact (from November 1999)*

Lauren Wood, *SoftQuad Software Inc., WG Chair*

Gavin Nicol, *Inso EPS (for DOM Level 1)*

Jonathan Robie, *Texcel Research and Software AG (for DOM Level 1)*

Mike Champion, *ArborText and Software AG (for DOM Level 1 from November 20, 1997)*

Steve Byrne, *JavaSoft (for DOM Level 1 until November 19, 1997)*

Copyright © 2000 W3C® (MIT, INRIA, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

Abstract

This specification defines the Document Object Model Level 2 Core, a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content and structure of documents. The Document Object Model Level 2 Core builds on the Document Object Model Level 1 Core.

The DOM Level 2 Core is made of a set of core interfaces to create and manipulate the structure and contents of a document. The Core also contains specialized interfaces dedicated to XML.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document has been produced as part of the W3C DOM Activity. The authors of this document are the DOM Working Group members. Different modules of the Document Object Model have different editors.

Please send general comments about this document to the public mailing list www-dom@w3.org. An archive is available at <http://lists.w3.org/Archives/Public/www-dom/>.

The English version of this specification is the only normative version. Information about translations of this document is available at <http://www.w3.org/2000/11/DOM-Level-2-translations>.

The list of known errors in this document is available at <http://www.w3.org/2000/11/DOM-Level-2-errata>

A list of current W3C Recommendations and other technical documents can be found at <http://www.w3.org/TR>.

Table of contents

Expanded Table of Contents3
Copyright Notice5
What is the Document Object Model?9
1. Document Object Model Core	15
Appendix A: Changes	67
Appendix B: Accessing code point boundaries	69
Appendix C: IDL Definitions	71
Appendix D: Java Language Binding	77
Appendix E: ECMAScript Language Binding	85
Appendix F: Acknowledgements	95
Glossary	97
References	101
Index	105

Expanded Table of Contents

Expanded Table of Contents3
Copyright Notice5
W3C Document Copyright Notice and License5
W3C Software Copyright Notice and License6
What is the Document Object Model?9
Introduction9
What the Document Object Model is9
What the Document Object Model is not	11
Where the Document Object Model came from	11
Entities and the DOM Core	12
Conformance	12
DOM Interfaces and DOM Implementations	13
1. Document Object Model Core	15
1.1. Overview of the DOM Core Interfaces	15
1.1.1. The DOM Structure Model	15
1.1.2. Memory Management	16
1.1.3. Naming Conventions	17
1.1.4. Inheritance vs. Flattened Views of the API	17
1.1.5. The DOMString type	17
1.1.6. The DOMTimeStamp type	18
1.1.7. String comparisons in the DOM	18
1.1.8. XML Namespaces	19
1.2. Fundamental Interfaces	20
1.3. Extended Interfaces	61
Appendix A: Changes	67
A.1. Changes between DOM Level 1 Core and DOM Level 2 Core	67
A.1.1. Changes to DOM Level 1 Core interfaces and exceptions	67
A.1.2. New features	68
Appendix B: Accessing code point boundaries	69
B.1. Introduction	69
B.2. Methods	69
Appendix C: IDL Definitions	71
Appendix D: Java Language Binding	77
Appendix E: ECMAScript Language Binding	85
Appendix F: Acknowledgements	95
F.1. Production Systems	95
Glossary	97
References	101
1. Normative references	101

Expanded Table of Contents

2. Informative references	101
Index	105

Copyright Notice

Copyright © 2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

This document is published under the W3C Document Copyright Notice and License [p.5] . The bindings within this document are published under the W3C Software Copyright Notice and License [p.6] . The software license requires "Notice of any changes or modifications to the W3C files, including the date changes were made." Consequently, modified versions of the DOM bindings must document that they do not conform to the W3C standard; in the case of the IDL definitions, the pragma prefix can no longer be 'w3c.org'; in the case of the Java Language binding, the package names can no longer be in the 'org.w3c' package.

W3C Document Copyright Notice and License

Note: This section is a copy of the W3C Document Notice and License and could be found at <http://www.w3.org/Consortium/Legal/copyright-documents-19990405>.

Copyright © 1994-2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

<http://www.w3.org/Consortium/Legal/>

Public documents on the W3C site are provided by the copyright holders under the following license. The software or Document Type Definitions (DTDs) associated with W3C specifications are governed by the Software Notice. By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on *ALL* copies of the document, or portions thereof, that you use:

1. A link or URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it doesn't exist, a notice of the form: "Copyright © [date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. <http://www.w3.org/Consortium/Legal/>" (Hypertext is preferred, but a textual representation is permitted.)
3. *If it exists*, the STATUS of the W3C document.

When space permits, inclusion of the full text of this **NOTICE** should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the Copyright FAQ) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

W3C Software Copyright Notice and License

Note: This section is a copy of the W3C Software Copyright Notice and License and could be found at <http://www.w3.org/Consortium/Legal/copyright-software-19980720>

Copyright © 1994-2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

<http://www.w3.org/Consortium/Legal/>

This W3C work (including software, documents, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and modify this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications, that you make:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers. If none exist, then a notice of the following form: "Copyright © [\$date-of-software] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. <http://www.w3.org/Consortium/Legal/>."

3. Notice of any changes or modifications to the W3C files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

What is the Document Object Model?

Editors

Philippe Le Hégaré, W3C
Lauren Wood, SoftQuad Software Inc., WG Chair
Jonathan Robie, Texcel (for DOM Level 1)

Introduction

The Document Object Model (DOM) is an application programming interface (*API* [p.97]) for valid *HTML* [p.98] and well-formed *XML* [p.99] documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions - in particular, the DOM *interfaces* [p.98] for the XML internal and external subsets have not yet been specified.

As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and *applications* [p.97] . The DOM is designed to be used with any programming language. In order to provide a precise, language-independent specification of the DOM interfaces, we have chosen to define the specifications in Object Management Group (OMG) IDL [OMGIDL], as defined in the CORBA 2.3.1 specification [CORBA]. In addition to the OMG IDL specification, we provide *language bindings* [p.98] for Java [Java] and ECMAScript [ECMAScript] (an industry-standard scripting language based on JavaScript [JavaScript] and JScript [JScript]).

Note: OMG IDL is used only as a language-independent and implementation-neutral way to specify *interfaces* [p.98] . Various other IDLs could have been used ([COM], [JavaIDL], [MIDL], ...). In general, IDLs are designed for specific computing environments. The Document Object Model can be implemented in any computing environment, and does not require the object binding runtimes generally associated with such IDLs.

What the Document Object Model is

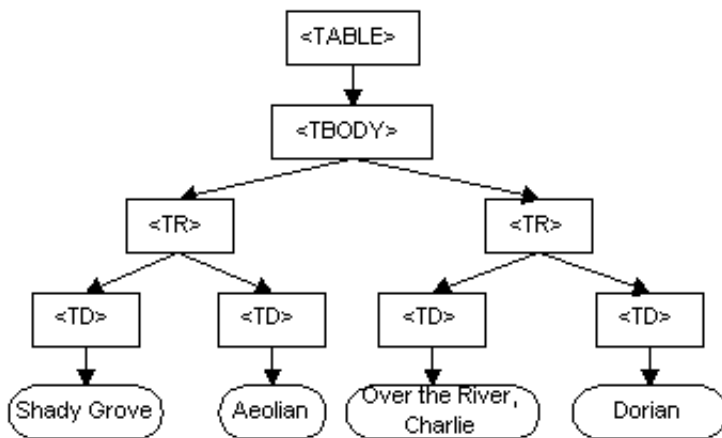
The DOM is a programming *API* [p.97] for documents. It is based on an object structure that closely resembles the structure of the documents it *models* [p.98] . For instance, consider this table, taken from an HTML document:

```

<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</TBODY>
</TABLE>

```

A graphical representation of the DOM of the example table is:



graphical representation of the DOM of the example table

In the DOM, documents have a logical structure which is very much like a tree; to be more precise, which is like a "forest" or "grove", which can contain more than one tree. Each document contains zero or one doctype nodes, one root element node, and zero or more comments or processing instructions; the root element serves as the root of the element tree for the document. However, the DOM does not specify that documents must be *implemented* as a tree or a grove, nor does it specify how the relationships among objects be implemented. The DOM is a logical model that may be implemented in any convenient manner. In this specification, we use the term *structure model* to describe the tree-like representation of a document. We also use the term "tree" when referring to the arrangement of those information items which can be reached by using "tree-walking" methods; (this does not include attributes). One important property of DOM structure models is *structural isomorphism*: if any two Document Object Model implementations are used to create a representation of the same document, they will create the same structure model, in accordance with the XML Information Set [Infoset].

Note: There may be some variations depending on the parser being used to build the DOM. For instance, the DOM may not contain whitespaces in element content if the parser discards them.

The name "Document Object Model" was chosen because it is an "*object model* [p.99] " in the traditional object oriented design sense: documents are modeled using objects, and the model encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed. In other words, the nodes in the above diagram do not represent a data structure, they represent objects, which have functions and identity. As an object model, the DOM identifies:

- the interfaces and objects used to represent and manipulate a document
- the semantics of these interfaces and objects - including both behavior and attributes
- the relationships and collaborations among these interfaces and objects

The structure of SGML documents has traditionally been represented by an abstract *data model* [p.97] , not by an object model. In an abstract *data model* [p.97] , the model is centered around the data. In object oriented programming languages, the data itself is encapsulated in objects that hide the data, protecting it from direct external manipulation. The functions associated with these objects determine how the objects may be manipulated, and they are part of the object model.

What the Document Object Model is not

This section is designed to give a more precise understanding of the DOM by distinguishing it from other systems that may seem to be like it.

- The Document Object Model is not a binary specification. DOM programs written in the same language binding will be source code compatible across platforms, but the DOM does not define any form of binary interoperability.
- The Document Object Model is not a way of persisting objects to XML or HTML. Instead of specifying how objects may be represented in XML, the DOM specifies how XML and HTML documents are represented as objects, so that they may be used in object oriented programs.
- The Document Object Model is not a set of data structures; it is an *object model* [p.99] that specifies interfaces. Although this document contains diagrams showing parent/child relationships, these are logical relationships defined by the programming interfaces, not representations of any particular internal data structures.
- The Document Object Model does not define what information in a document is relevant or how information in a document is structured. For XML, this is specified by the W3C XML Information Set [Infoset]. The DOM is simply an *API* [p.97] to this information set.
- The Document Object Model, despite its name, is not a competitor to the Component Object Model (COM). COM, like CORBA, is a language independent way to specify interfaces and objects; the DOM is a set of interfaces and objects designed for managing HTML and XML documents. The DOM may be implemented using language-independent systems like COM or CORBA; it may also be implemented using language-specific bindings like the Java or ECMAScript bindings specified in this document.

Where the Document Object Model came from

The DOM originated as a specification to allow JavaScript scripts and Java programs to be portable among Web browsers. "Dynamic HTML" was the immediate ancestor of the Document Object Model, and it was originally thought of largely in terms of browsers. However, when the DOM Working Group was formed at W3C, it was also joined by vendors in other domains, including HTML or XML editors and document repositories. Several of these vendors had worked with SGML before XML was developed; as a result, the DOM has been influenced by SGML Groves and the HyTime standard. Some of these vendors had also developed their own object models for documents in order to provide an API for SGML/XML editors or document repositories, and these object models have also influenced the DOM.

Entities and the DOM Core

In the fundamental DOM interfaces, there are no objects representing entities. Numeric character references, and references to the pre-defined entities in HTML and XML, are replaced by the single character that makes up the entity's replacement. For example, in:

```
<p>This is a dog &amp; a cat</p>
```

the "&" will be replaced by the character "&", and the text in the P element will form a single continuous sequence of characters. Since numeric character references and pre-defined entities are not recognized as such in CDATA sections, or in the SCRIPT and STYLE elements in HTML, they are not replaced by the single character they appear to refer to. If the example above were enclosed in a CDATA section, the "&" would not be replaced by "&"; neither would the <p> be recognized as a start tag. The representation of general entities, both internal and external, are defined within the extended (XML) interfaces of DOM Level 1 [DOM Level 1].

Note: When a DOM representation of a document is serialized as XML or HTML text, applications will need to check each character in text data to see if it needs to be escaped using a numeric or pre-defined entity. Failing to do so could result in invalid HTML or XML. Also, *implementations* [p.98] should be aware of the fact that serialization into a character encoding ("charset") that does not fully cover ISO 10646 may fail if there are characters in markup or CDATA sections that are not present in the encoding.

Conformance

This section explains the different levels of conformance to DOM Level 2. DOM Level 2 consists of 14 modules. It is possible to conform to DOM Level 2, or to a DOM Level 2 module.

An implementation is DOM Level 2 conformant if it supports the Core module defined in this document (see Fundamental Interfaces [p.20]). An implementation conforms to a DOM Level 2 module if it supports all the interfaces for that module and the associated semantics.

Here is the complete list of DOM Level 2.0 modules and the features used by them. Feature names are case-insensitive.

Core module

defines the feature "*Core*" [p.20] .

XML module

defines the feature "*XML*" [p.61] .

HTML module

defines the feature "*HTML*". (see [DOM Level 2 HTML]).

Note: At time of publication, this DOM Level 2 module is not yet a W3C Recommendation.

Views module

defines the feature "*Views*" in [DOM Level 2 Views].

Style Sheets module

defines the feature "*StyleSheets*" in [DOM Level 2 Style Sheets].

CSS module

defines the feature "*CSS*" in [DOM Level 2 CSS].

CSS2 module

defines the feature "*CSS2*" in [DOM Level 2 CSS].

Events module

defines the feature "*Events*" in [DOM Level 2 Events].

User interface Events module

defines the feature "*UIEvents*" in [DOM Level 2 Events].

Mouse Events module

defines the feature "*MouseEvents*" in [DOM Level 2 Events].

Mutation Events module

defines the feature "*MutationEvents*" in [DOM Level 2 Events].

HTML Events module

defines the feature "*HTMLEvents*" in [DOM Level 2 Events].

Range module

defines the feature "*Range*" in [DOM Level 2 Range].

Traversal module

defines the feature "*Traversal*" in [DOM Level 2 Traversal].

A DOM implementation must not return "*true*" to the `hasFeature(feature, version)` *method* [p.98] of the `DOMImplementation` [p.22] interface for that feature unless the implementation conforms to that module. The `version` number for all features used in DOM Level 2.0 is "*2.0*".

DOM Interfaces and DOM Implementations

The DOM specifies interfaces which may be used to manage XML or HTML documents. It is important to realize that these interfaces are an abstraction - much like "abstract base classes" in C++, they are a means of specifying a way to access and manipulate an application's internal representation of a document. Interfaces do not imply a particular concrete implementation. Each DOM application is free to maintain documents in any convenient representation, as long as the interfaces shown in this specification are supported. Some DOM implementations will be existing programs that use the DOM interfaces to access software written long before the DOM specification existed. Therefore, the DOM is designed to avoid implementation dependencies; in particular,

1. Attributes defined in the IDL do not imply concrete objects which must have specific data members - in the language bindings, they are translated to a pair of get()/set() functions, not to a data member. Read-only attributes have only a get() function in the language bindings.
2. DOM applications may provide additional interfaces and objects not found in this specification and still be considered DOM conformant.
3. Because we specify interfaces and not the actual objects that are to be created, the DOM cannot know what constructors to call for an implementation. In general, DOM users call the createX() methods on the Document class to create document structures, and DOM implementations create their own internal representations of these structures in their implementations of the createX() functions.

The Level 1 interfaces were extended to provide both Level 1 and Level 2 functionality.

DOM implementations in languages other than Java or ECMAScript may choose bindings that are appropriate and natural for their language and run time environment. For example, some systems may need to create a Document2 class which inherits from Document and contains the new methods and attributes.

DOM Level 2 does not specify multithreading mechanisms.

1. Document Object Model Core

Editors

Arnaud Le Hors, IBM
 Gavin Nicol, Inso EPS (for DOM Level 1)
 Lauren Wood, SoftQuad, Inc. (for DOM Level 1)
 Mike Champion, ArborText (for DOM Level 1 from November 20, 1997)
 Steve Byrne, JavaSoft (for DOM Level 1 until November 19, 1997)

1.1. Overview of the DOM Core Interfaces

This section defines a set of objects and interfaces for accessing and manipulating document objects. The functionality specified in this section (the *Core* functionality) is sufficient to allow software developers and web script authors to access and manipulate parsed HTML and XML content inside conforming products. The DOM Core API also allows creation and population of a `Document` [p.25] object using only DOM API calls; loading a `Document` and saving it persistently is left to the product that implements the DOM API.

1.1.1. The DOM Structure Model

The DOM presents documents as a hierarchy of `Node` [p.34] objects that also implement other, more specialized interfaces. Some types of nodes may have *child* [p.97] nodes of various types, and others are leaf nodes that cannot have anything below them in the document structure. For XML and HTML, the node types, and which node types they may have as children, are as follows:

- `Document` [p.25] -- `Element` [p.52] (maximum of one), `ProcessingInstruction` [p.66], `Comment` [p.61], `DocumentType` [p.62] (maximum of one)
- `DocumentFragment` [p.24] -- `Element` [p.52], `ProcessingInstruction` [p.66], `Comment` [p.61], `Text` [p.60], `CDATASection` [p.62], `EntityReference` [p.65]
- `DocumentType` [p.62] -- no children
- `EntityReference` [p.65] -- `Element` [p.52], `ProcessingInstruction` [p.66], `Comment` [p.61], `Text` [p.60], `CDATASection` [p.62], `EntityReference`
- `Element` [p.52] -- `Element`, `Text` [p.60], `Comment` [p.61], `ProcessingInstruction` [p.66], `CDATASection` [p.62], `EntityReference` [p.65]
- `Attr` [p.51] -- `Text` [p.60], `EntityReference` [p.65]
- `ProcessingInstruction` [p.66] -- no children
- `Comment` [p.61] -- no children
- `Text` [p.60] -- no children
- `CDATASection` [p.62] -- no children
- `Entity` [p.64] -- `Element` [p.52], `ProcessingInstruction` [p.66], `Comment` [p.61], `Text` [p.60], `CDATASection` [p.62], `EntityReference` [p.65]
- `Notation` [p.64] -- no children

The DOM also specifies a `NodeList` [p.43] interface to handle ordered lists of `Nodes` [p.34], such as the children of a `Node` [p.34], or the *elements* [p.98] returned by the `getElementsByTagName` method of the `Element` [p.52] interface, and also a `NamedNodeMap` [p.44] interface to handle unordered sets of nodes referenced by their name attribute, such as the attributes of an `Element`. `NodeList` [p.43] and `NamedNodeMap` [p.44] objects in the DOM are *live*; that is, changes to the underlying document structure are reflected in all relevant `NodeList` and `NamedNodeMap` objects. For example, if a DOM user gets a `NodeList` object containing the children of an `Element` [p.52], then subsequently adds more children to that *element* [p.98] (or removes children, or modifies them), those changes are automatically reflected in the `NodeList`, without further action on the user's part. Likewise, changes to a `Node` [p.34] in the tree are reflected in all references to that `Node` in `NodeList` and `NamedNodeMap` objects.

Finally, the interfaces `Text` [p.60], `Comment` [p.61], and `CDATASection` [p.62] all inherit from the `CharacterData` [p.47] interface.

1.1.2. Memory Management

Most of the APIs defined by this specification are *interfaces* rather than classes. That means that an implementation need only expose methods with the defined names and specified operation, not implement classes that correspond directly to the interfaces. This allows the DOM APIs to be implemented as a thin veneer on top of legacy applications with their own data structures, or on top of newer applications with different class hierarchies. This also means that ordinary constructors (in the Java or C++ sense) cannot be used to create DOM objects, since the underlying objects to be constructed may have little relationship to the DOM interfaces. The conventional solution to this in object-oriented design is to define *factory* methods that create instances of objects that implement the various interfaces. Objects implementing some interface "X" are created by a "createX()" method on the `Document` [p.25] interface; this is because all DOM objects live in the context of a specific `Document`.

The DOM Level 2 API does *not* define a standard way to create `DOMImplementation` [p.22] objects; DOM implementations must provide some proprietary way of bootstrapping these DOM interfaces, and then all other objects can be built from there.

The Core DOM APIs are designed to be compatible with a wide range of languages, including both general-user scripting languages and the more challenging languages used mostly by professional programmers. Thus, the DOM APIs need to operate across a variety of memory management philosophies, from language bindings that do not expose memory management to the user at all, through those (notably Java) that provide explicit constructors but provide an automatic garbage collection mechanism to automatically reclaim unused memory, to those (especially C/C++) that generally require the programmer to explicitly allocate object memory, track where it is used, and explicitly free it for re-use. To ensure a consistent API across these platforms, the DOM does not address memory management issues at all, but instead leaves these for the implementation. Neither of the explicit language bindings defined by the DOM API (for *ECMAScript* [p.98] and Java) require any memory management methods, but DOM bindings for other languages (especially C or C++) may require such support. These extensions will be the responsibility of those adapting the DOM API to a specific language, not the DOM Working Group.

1.1.3. Naming Conventions

While it would be nice to have attribute and method names that are short, informative, internally consistent, and familiar to users of similar APIs, the names also should not clash with the names in legacy APIs supported by DOM implementations. Furthermore, both OMG IDL and ECMAScript have significant limitations in their ability to disambiguate names from different namespaces that make it difficult to avoid naming conflicts with short, familiar names. So, DOM names tend to be long and descriptive in order to be unique across all environments.

The Working Group has also attempted to be internally consistent in its use of various terms, even though these may not be common distinctions in other APIs. For example, the DOM API uses the method name "remove" when the method changes the structural model, and the method name "delete" when the method gets rid of something inside the structure model. The thing that is deleted is not returned. The thing that is removed may be returned, when it makes sense to return it.

1.1.4. Inheritance vs. Flattened Views of the API

The DOM Core *APIs* [p.97] present two somewhat different sets of interfaces to an XML/HTML document: one presenting an "object oriented" approach with a hierarchy of *inheritance* [p.98] , and a "simplified" view that allows all manipulation to be done via the *Node* [p.34] interface without requiring casts (in Java and other C-like languages) or query interface calls in *COM* [p.97] environments. These operations are fairly expensive in Java and COM, and the DOM may be used in performance-critical environments, so we allow significant functionality using just the *Node* interface. Because many other users will find the *inheritance* [p.98] hierarchy easier to understand than the "everything is a *Node*" approach to the DOM, we also support the full higher-level interfaces for those who prefer a more object-oriented *API* [p.97] .

In practice, this means that there is a certain amount of redundancy in the *API* [p.97] . The Working Group considers the "*inheritance* [p.98] " approach the primary view of the API, and the full set of functionality on *Node* [p.34] to be "extra" functionality that users may employ, but that does not eliminate the need for methods on other interfaces that an object-oriented analysis would dictate. (Of course, when the O-O analysis yields an attribute or method that is identical to one on the *Node* interface, we don't specify a completely redundant one.) Thus, even though there is a generic *nodeName* attribute on the *Node* interface, there is still a *tagName* attribute on the *Element* [p.52] interface; these two attributes must contain the same value, but the it is worthwhile to support both, given the different constituencies the DOM *API* [p.97] must satisfy.

1.1.5. The DOMString type

To ensure interoperability, the DOM specifies the following:



Type Definition *DOMString*

A *DOMString* [p.17] is a sequence of *16-bit units* [p.97] .

IDL Definition

```
valuetype DOMString sequence<unsigned short>;
```

- Applications must encode DOMString [p.17] using UTF-16 (defined in [Unicode] and Amendment 1 of [ISO/IEC 10646]).

The UTF-16 encoding was chosen because of its widespread industry practice. Note that for both HTML and XML, the document character set (and therefore the notation of numeric character references) is based on UCS [ISO-10646]. A single numeric character reference in a source document may therefore in some cases correspond to two 16-bit units in a DOMString [p.17] (a high surrogate and a low surrogate).

Note: Even though the DOM defines the name of the string type to be DOMString [p.17], bindings may use different names. For example for Java, DOMString is bound to the String type because it also uses UTF-16 as its encoding.

Note: As of August 2000, the OMG IDL specification ([OMGIDL]) included a wstring type. However, that definition did not meet the interoperability criteria of the DOM API [p.97] since it relied on negotiation to decide the width and encoding of a character.

1.1.6. The DOMTimeStamp type

To ensure interoperability, the DOM specifies the following:

-

Type Definition *DOMTimeStamp*

A DOMTimeStamp [p.18] represents a number of milliseconds.

IDL Definition

```
typedef unsigned long long DOMTimeStamp;
```

- **Note:** Even though the DOM uses the type DOMTimeStamp [p.18], bindings may use different types. For example for Java, DOMTimeStamp is bound to the long type. In ECMAScript, TimeStamp is bound to the Date type because the range of the integer type is too small.

1.1.7. String comparisons in the DOM

The DOM has many interfaces that imply string matching. HTML processors generally assume an uppercase (less often, lowercase) normalization of names for such things as *elements* [p.98], while XML is explicitly case sensitive. For the purposes of the DOM, string matching is performed purely by binary *comparison* [p.99] of the *16-bit units* [p.97] of the DOMString [p.17]. In addition, the DOM assumes that any case normalizations take place in the processor, *before* the DOM structures are built.

Note: Besides case folding, there are additional normalizations that can be applied to text. The W3C I18N Working Group is in the process of defining exactly which normalizations are necessary, and where they should be applied. The W3C I18N Working Group expects to require early normalization, which means that data read into the DOM is assumed to already be normalized. The DOM and applications built on top

of it in this case only have to assure that text remains normalized when being changed. For further details, please see [Charmod].

1.1.8. XML Namespaces

The DOM Level 2 supports XML namespaces [Namespaces] by augmenting several interfaces of the DOM Level 1 Core to allow creating and manipulating *elements* [p.98] and attributes associated to a namespace.

As far as the DOM is concerned, special attributes used for declaring *XML namespaces* [p.100] are still exposed and can be manipulated just like any other attribute. However, nodes are permanently bound to *namespace URIs* [p.99] as they get created. Consequently, moving a node within a document, using the DOM, in no case results in a change of its *namespace prefix* [p.99] or namespace URI. Similarly, creating a node with a namespace prefix and namespace URI, or changing the namespace prefix of a node, does not result in any addition, removal, or modification of any special attributes for declaring the appropriate XML namespaces. Namespace validation is not enforced; the DOM application is responsible. In particular, since the mapping between prefixes and namespace URIs is not enforced, in general, the resulting document cannot be serialized naively. For example, applications may have to declare every namespace in use when serializing a document.

DOM Level 2 doesn't perform any URI normalization or canonicalization. The URIs given to the DOM are assumed to be valid (e.g., characters such as whitespaces are properly escaped), and no lexical checking is performed. Absolute URI references are treated as strings and *compared literally* [p.99]. How relative namespace URI references are treated is undefined. To ensure interoperability only absolute namespace URI references (i.e., URI references beginning with a scheme name and a colon) should be used. Note that because the DOM does no lexical checking, the empty string will be treated as a real namespace URI in DOM Level 2 methods. Applications must use the value `null` as the namespaceURI parameter for methods if they wish to have no namespace.

Note: In the DOM, all namespace declaration attributes are *by definition* bound to the namespace URI: "http://www.w3.org/2000/xmlns/". These are the attributes whose *namespace prefix* [p.99] or *qualified name* [p.99] is "xmlns". Although, at the time of writing, this is not part of the XML Namespaces specification [Namespaces], it is planned to be incorporated in a future revision.

In a document with no namespaces, the *child* [p.97] list of an *EntityReference* [p.65] node is always the same as that of the corresponding *Entity* [p.64]. This is not true in a document where an entity contains unbound *namespace prefixes* [p.99]. In such a case, the *descendants* [p.97] of the corresponding *EntityReference* nodes may be bound to different *namespace URIs* [p.99], depending on where the entity references are. Also, because, in the DOM, nodes always remain bound to the same namespace URI, moving such *EntityReference* nodes can lead to documents that cannot be serialized. This is also true when the DOM Level 1 method *createEntityReference* of the *Document* [p.25] interface is used to create entity references that correspond to such entities, since the *descendants* [p.97] of the returned *EntityReference* are unbound. The DOM Level 2 does not support any mechanism to resolve namespace prefixes. For all of these reasons, use of such entities and entity references should be avoided or used with extreme care. A future Level of the DOM may include some additional support for handling these.

The new methods, such as `createElementNS` and `createAttributeNS` of the Document [p.25] interface, are meant to be used by namespace aware applications. Simple applications that do not use namespaces can use the DOM Level 1 methods, such as `createElement` and `createAttribute`. Elements and attributes created in this way do not have any namespace prefix, namespace URI, or local name.

Note: DOM Level 1 methods are namespace ignorant. Therefore, while it is safe to use these methods when not dealing with namespaces, using them and the new ones at the same time should be avoided. DOM Level 1 methods solely identify attribute nodes by their `nodeName`. On the contrary, the DOM Level 2 methods related to namespaces, identify attribute nodes by their `namespaceURI` and `localName`. Because of this fundamental difference, mixing both sets of methods can lead to unpredictable results. In particular, using `setAttributeNS`, an *element* [p.98] may have two attributes (or more) that have the same `nodeName`, but different `namespaceURIs`. Calling `getAttribute` with that `nodeName` could then return any of those attributes. The result depends on the implementation. Similarly, using `setAttributeNode`, one can set two attributes (or more) that have different `nodeNames` but the same `prefix` and `namespaceURI`. In this case `getAttributeNodeNS` will return either attribute, in an implementation dependent manner. The only guarantee in such cases is that all methods that access a named item by its `nodeName` will access the same item, and all methods which access a node by its URI and local name will access the same node. For instance, `setAttribute` and `setAttributeNS` affect the node that `getAttribute` and `getAttributeNS`, respectively, return.

1.2. Fundamental Interfaces

The interfaces within this section are considered *fundamental*, and must be fully implemented by all conforming implementations of the DOM, including all HTML DOM implementations [DOM Level 2 HTML], unless otherwise specified.

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` [p.22] interface with parameter values "Core" and "2.0" (respectively) to determine whether or not this module is supported by the implementation. Any implementation that conforms to DOM Level 2 or a DOM Level 2 module must conform to the Core module. Please refer to additional information about *conformance* in this specification.

Exception *DOMException*

DOM operations only raise exceptions in "exceptional" circumstances, i.e., when an operation is impossible to perform (either for logical reasons, because data is lost, or because the implementation has become unstable). In general, DOM methods return specific error values in ordinary processing situations, such as out-of-bound errors when using `NodeList` [p.43] .

Implementations should raise other exceptions under other circumstances. For example, implementations should raise an implementation-dependent exception if a `null` argument is passed.

Some languages and object systems do not support the concept of exceptions. For such systems, error conditions may be indicated using native error reporting mechanisms. For some bindings, for example, methods may return error codes similar to those listed in the corresponding method descriptions.

IDL Definition

```

exception DOMException {
    unsigned short    code;
};
// ExceptionCode
const unsigned short    INDEX_SIZE_ERR                = 1;
const unsigned short    DOMSTRING_SIZE_ERR            = 2;
const unsigned short    HIERARCHY_REQUEST_ERR        = 3;
const unsigned short    WRONG_DOCUMENT_ERR           = 4;
const unsigned short    INVALID_CHARACTER_ERR        = 5;
const unsigned short    NO_DATA_ALLOWED_ERR          = 6;
const unsigned short    NO_MODIFICATION_ALLOWED_ERR   = 7;
const unsigned short    NOT_FOUND_ERR                 = 8;
const unsigned short    NOT_SUPPORTED_ERR             = 9;
const unsigned short    INUSE_ATTRIBUTE_ERR           = 10;
// Introduced in DOM Level 2:
const unsigned short    INVALID_STATE_ERR             = 11;
// Introduced in DOM Level 2:
const unsigned short    SYNTAX_ERR                   = 12;
// Introduced in DOM Level 2:
const unsigned short    INVALID_MODIFICATION_ERR      = 13;
// Introduced in DOM Level 2:
const unsigned short    NAMESPACE_ERR                = 14;
// Introduced in DOM Level 2:
const unsigned short    INVALID_ACCESS_ERR            = 15;

```

Definition group *ExceptionCode*

An integer indicating the type of error generated.

Note: Other numeric codes are reserved for W3C for possible future use.

Defined Constants

DOMSTRING_SIZE_ERR

If the specified range of text does not fit into a DOMString

HIERARCHY_REQUEST_ERR

If any node is inserted somewhere it doesn't belong

INDEX_SIZE_ERR

If index or size is negative, or greater than the allowed value

INUSE_ATTRIBUTE_ERR

If an attempt is made to add an attribute that is already in use elsewhere

INVALID_ACCESS_ERR, introduced in **DOM Level 2**.

If a parameter or an operation is not supported by the underlying object.

INVALID_CHARACTER_ERR

If an invalid or illegal character is specified, such as in a name. See *production 2* in the XML specification for the definition of a legal character, and *production 5* for the definition of a legal name character.

INVALID_MODIFICATION_ERR, introduced in **DOM Level 2**.

If an attempt is made to modify the type of the underlying object.

INVALID_STATE_ERR, introduced in **DOM Level 2**.

If an attempt is made to use an object that is not, or is no longer, usable.

NAMESPACE_ERR, introduced in **DOM Level 2**.

If an attempt is made to create or change an object in a way which is incorrect with regard to namespaces.

NOT_FOUND_ERR

If an attempt is made to reference a node in a context where it does not exist

NOT_SUPPORTED_ERR

If the implementation does not support the requested type of object or operation.

NO_DATA_ALLOWED_ERR

If data is specified for a node which does not support data

NO_MODIFICATION_ALLOWED_ERR

If an attempt is made to modify an object where modifications are not allowed

SYNTAX_ERR, introduced in **DOM Level 2**.

If an invalid or illegal string is specified.

WRONG_DOCUMENT_ERR

If a node is used in a different document than the one that created it (that doesn't support it)

Interface *DOMImplementation*

The *DOMImplementation* interface provides a number of methods for performing operations that are independent of any particular instance of the document object model.

IDL Definition

```
interface DOMImplementation {
    boolean          hasFeature(in DOMString feature,
                                in DOMString version);

    // Introduced in DOM Level 2:
    DocumentType     createDocumentType(in DOMString qualifiedName,
                                        in DOMString publicId,
                                        in DOMString systemId)
                                raises(DOMException);

    // Introduced in DOM Level 2:
    Document         createDocument(in DOMString namespaceURI,
                                    in DOMString qualifiedName,
                                    in DocumentType doctype)
                                raises(DOMException);
};
```

Methods

createDocument introduced in **DOM Level 2**

Creates an XML Document [p.25] object of the specified type with its document element. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString [p.17]

The *namespace URI* [p.99] of the document element to create.

qualifiedName of type DOMString

The *qualified name* [p.99] of the document element to be created.

doctype of type `DocumentType` [p.62]

The type of document to be created or `null`.

When doctype is not `null`, its `Node.ownerDocument` [p.38] attribute is set to the document being created.

Return Value

`Document` [p.25] A new `Document` object.

Exceptions

`DOMException` [p.20] `INVALID_CHARACTER_ERR`: Raised if the specified qualified name contains an illegal character.

`NAMESPACE_ERR`: Raised if the `qualifiedName` is malformed, if the `qualifiedName` has a prefix and the `namespaceURI` is `null`, or if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from "http://www.w3.org/XML/1998/namespace" [Namespaces].

`WRONG_DOCUMENT_ERR`: Raised if doctype has already been used with a different document or was created from a different implementation.

`createDocumentType` introduced in **DOM Level 2**

Creates an empty `DocumentType` [p.62] node. Entity declarations and notations are not made available. Entity reference expansions and default attribute additions do not occur. It is expected that a future version of the DOM will provide a way for populating a `DocumentType`.

HTML-only DOM implementations do not need to implement this method.

Parameters

`qualifiedName` of type `DOMString` [p.17]

The *qualified name* [p.99] of the document type to be created.

`publicId` of type `DOMString`

The external subset public identifier.

`systemId` of type `DOMString`

The external subset system identifier.

Return Value

`DocumentType` [p.62] A new `DocumentType` node with `Node.ownerDocument` [p.38] set to `null`.

Exceptions

`DOMException` [p.20] `INVALID_CHARACTER_ERR`: Raised if the specified qualified name contains an illegal character.

`NAMESPACE_ERR`: Raised if the `qualifiedName` is malformed.

hasFeature

Test if the DOM implementation implements a specific feature.

Parameters

`feature` of type `DOMString` [p.17]

The name of the feature to test (case-insensitive). The values used by DOM features are defined throughout the DOM Level 2 specifications and listed in the Conformance [p.12] section. The name must be an *XML name* [p.99]. To avoid possible conflicts, as a convention, names referring to features defined outside the DOM specification should be made unique by reversing the name of the Internet domain name of the person (or the organization that the person belongs to) who defines the feature, component by component, and using this as a prefix. For instance, the W3C SVG Working Group defines the feature "org.w3c.dom.svg".

`version` of type `DOMString`

This is the version number of the feature to test. In Level 2, the string can be either "2.0" or "1.0". If the version is not specified, supporting any version of the feature causes the method to return `true`.

Return Value

`boolean` `true` if the feature is implemented in the specified version, `false` otherwise.

No Exceptions**Interface *DocumentFragment***

`DocumentFragment` is a "lightweight" or "minimal" `Document` [p.25] object. It is very common to want to be able to extract a portion of a document's tree or to create a new fragment of a document. Imagine implementing a user command like cut or rearranging a document by moving fragments around. It is desirable to have an object which can hold such fragments and it is quite natural to use a `Node` for this purpose. While it is true that a `Document` object could fulfill this role, a `Document` object can potentially be a heavyweight object, depending on the underlying implementation. What is really needed for this is a very lightweight object. `DocumentFragment` is such an object.

Furthermore, various operations -- such as inserting nodes as children of another `Node` [p.34] -- may take `DocumentFragment` objects as arguments; this results in all the child nodes of the `DocumentFragment` being moved to the child list of this node.

The children of a `DocumentFragment` node are zero or more nodes representing the tops of any sub-trees defining the structure of the document. `DocumentFragment` nodes do not need to be *well-formed XML documents* [p.99] (although they do need to follow the rules imposed upon

well-formed XML parsed entities, which can have multiple top nodes). For example, a `DocumentFragment` might have only one child and that child node could be a `Text` [p.60] node. Such a structure model represents neither an HTML document nor a well-formed XML document.

When a `DocumentFragment` is inserted into a `Document` [p.25] (or indeed any other `Node` [p.34] that may take children) the children of the `DocumentFragment` and not the `DocumentFragment` itself are inserted into the `Node`. This makes the `DocumentFragment` very useful when the user wishes to create nodes that are *siblings* [p.99]; the `DocumentFragment` acts as the parent of these nodes so that the user can use the standard methods from the `Node` interface, such as `insertBefore` and `appendChild`.

IDL Definition

```
interface DocumentFragment : Node {
};
```

Interface *Document*

The `Document` interface represents the entire HTML or XML document. Conceptually, it is the *root* [p.99] of the document tree, and provides the primary access to the document's data.

Since elements, text nodes, comments, processing instructions, etc. cannot exist outside the context of a `Document`, the `Document` interface also contains the factory methods needed to create these objects. The `Node` [p.34] objects created have a `ownerDocument` attribute which associates them with the `Document` within whose context they were created.

IDL Definition

```
interface Document : Node {
  readonly attribute DocumentType      doctype;
  readonly attribute DOMImplementation implementation;
  readonly attribute Element           documentElement;
  Element                             createElement(in DOMString tagName)
                                      raises(DOMException);
  DocumentFragment                    createDocumentFragment();
  Text                                createTextNode(in DOMString data);
  Comment                             createComment(in DOMString data);
  CDATASection                        createCDATASection(in DOMString data)
                                      raises(DOMException);
  ProcessingInstruction createProcessingInstruction(in DOMString target,
                                                  in DOMString data)
                                      raises(DOMException);
  Attr                                createAttribute(in DOMString name)
                                      raises(DOMException);
  EntityReference                     createEntityReference(in DOMString name)
                                      raises(DOMException);
  NodeList                            getElementsByTagName(in DOMString tagname);
  // Introduced in DOM Level 2:
  Node                                importNode(in Node importedNode,
                                                  in boolean deep)
                                      raises(DOMException);
  // Introduced in DOM Level 2:
  Element                             createElementNS(in DOMString namespaceURI,
```

```

                                in DOMString qualifiedName)
                                raises(DOMException);

// Introduced in DOM Level 2:
Attr                            createAttributeNS(in DOMString namespaceURI,
                                                in DOMString qualifiedName)
                                                raises(DOMException);

// Introduced in DOM Level 2:
NodeList                        getElementsByTagNameNS(in DOMString namespaceURI,
                                                in DOMString localName);

// Introduced in DOM Level 2:
Element                        getElementById(in DOMString elementId);
};

```

Attributes

doctype of type `DocumentType` [p.62] , readonly

The Document Type Declaration (see `DocumentType` [p.62]) associated with this document. For HTML documents as well as XML documents without a document type declaration this returns null. The DOM Level 2 does not support editing the Document Type Declaration. `doctype` cannot be altered in any way, including through the use of methods inherited from the `Node` [p.34] interface, such as `insertNode` or `removeNode`.

documentElement of type `Element` [p.52] , readonly

This is a *convenience* [p.97] attribute that allows direct access to the child node that is the root element of the document. For HTML documents, this is the element with the tagName "HTML".

implementation of type `DOMImplementation` [p.22] , readonly

The `DOMImplementation` [p.22] object that handles this document. A DOM application may use objects from multiple implementations.

Methods

`createAttribute`

Creates an `Attr` [p.51] of the given name. Note that the `Attr` instance can then be set on an `Element` [p.52] using the `setAttributeNode` method.

To create an attribute with a qualified name and namespace URI, use the `createAttributeNS` method.

Parameters

name of type `DOMString` [p.17]

The name of the attribute.

Return Value

<code>Attr</code> [p.51]	A new <code>Attr</code> object with the <code>nodeName</code> attribute set to name, and <code>localName</code> , <code>prefix</code> , and <code>namespaceURI</code> set to null. The value of the attribute is the empty string.
-----------------------------	--

Exceptions

<code>DOMException</code> [p.20]	<code>INVALID_CHARACTER_ERR</code> : Raised if the specified name contains an illegal character.
-------------------------------------	--

createAttributeNS introduced in **DOM Level 2**

Creates an attribute of the given qualified name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString [p.17]

The *namespace URI* [p.99] of the attribute to create.

qualifiedName of type DOMString

The *qualified name* [p.99] of the attribute to instantiate.

Return Value

Attr [p.51] A new Attr object with the following attributes:

Attribute	Value
Node.nodeName [p.37]	qualifiedName
Node.namespaceURI [p.37]	namespaceURI
Node.prefix [p.38]	prefix, extracted from qualifiedName, or null if there is no prefix
Node.localName [p.37]	<i>local name</i> , extracted from qualifiedName
Attr.name [p.52]	qualifiedName
Node.nodeValue [p.37]	the empty string

Exceptions

DOMException [p.20] **INVALID_CHARACTER_ERR**: Raised if the specified qualified name contains an illegal character.

NAMESPACE_ERR: Raised if the qualifiedName is malformed, if the qualifiedName has a prefix and the namespaceURI is null, if the qualifiedName has a prefix that is "xml" and the namespaceURI is different from "http://www.w3.org/XML/1998/namespace", or if the qualifiedName is "xmlns" and the namespaceURI is different from "http://www.w3.org/2000/xmlns/".

createCDATASection

Creates a CDATASection [p.62] node whose value is the specified string.

Parameters

data of type `DOMString` [p.17]

The data for the `CDATASection` [p.62] contents.

Return Value

`CDATASection` [p.62] The new `CDATASection` object.

Exceptions

<code>DOMException</code> [p.20]	<code>NOT_SUPPORTED_ERR</code> : Raised if this document is an HTML document.
-------------------------------------	---

`createComment`

Creates a `Comment` [p.61] node given the specified string.

Parameters

data of type `DOMString` [p.17]

The data for the node.

Return Value

`Comment` [p.61] The new `Comment` object.

No Exceptions

`createDocumentFragment`

Creates an empty `DocumentFragment` [p.24] object.

Return Value

`DocumentFragment` [p.24] A new `DocumentFragment`.

No Parameters

No Exceptions

`createElement`

Creates an element of the type specified. Note that the instance returned implements the `Element` [p.52] interface, so attributes can be specified directly on the returned object.

In addition, if there are known attributes with default values, `Attr` [p.51] nodes representing them are automatically created and attached to the element.

To create an element with a qualified name and namespace URI, use the `createElementNS` method.

Parameters

tagName of type `DOMString` [p.17]

The name of the element type to instantiate. For XML, this is case-sensitive. For

HTML, the `tagName` parameter may be provided in any case, but it must be mapped to the canonical uppercase form by the DOM implementation.

Return Value

`Element` [p.52] A new `Element` object with the `nodeName` attribute set to `tagName`, and `localName`, `prefix`, and `namespaceURI` set to `null`.

Exceptions

`DOMException` [p.20] `INVALID_CHARACTER_ERR`: Raised if the specified name contains an illegal character.

`createElementNS` introduced in **DOM Level 2**

Creates an element of the given qualified name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters

`namespaceURI` of type `DOMString` [p.17]

The *namespace URI* [p.99] of the element to create.

`qualifiedName` of type `DOMString`

The *qualified name* [p.99] of the element type to instantiate.

Return Value

`Element` [p.52] A new `Element` object with the following attributes:

Attribute	Value
<code>Node.nodeName</code> [p.37]	<code>qualifiedName</code>
<code>Node.namespaceURI</code> [p.37]	<code>namespaceURI</code>
<code>Node.prefix</code> [p.38]	prefix, extracted from <code>qualifiedName</code> , or <code>null</code> if there is no prefix
<code>Node.localName</code> [p.37]	<i>local name</i> , extracted from <code>qualifiedName</code>
<code>Element.tagName</code> [p.54]	<code>qualifiedName</code>

Exceptions

DOMException [p.20]	<p>INVALID_CHARACTER_ERR: Raised if the specified qualified name contains an illegal character.</p> <p>NAMESPACE_ERR: Raised if the <code>qualifiedName</code> is malformed, if the <code>qualifiedName</code> has a prefix and the <code>namespaceURI</code> is null, or if the <code>qualifiedName</code> has a prefix that is "xml" and the <code>namespaceURI</code> is different from "http://www.w3.org/XML/1998/namespace" [Namespaces].</p>
------------------------	---

createEntityReference

Creates an `EntityReference` [p.65] object. In addition, if the referenced entity is known, the child list of the `EntityReference` node is made the same as that of the corresponding `Entity` [p.64] node.

Note: If any descendant of the `Entity` [p.64] node has an unbound *namespace prefix* [p.99], the corresponding descendant of the created `EntityReference` [p.65] node is also unbound; (its `namespaceURI` is null). The DOM Level 2 does not support any mechanism to resolve namespace prefixes.

Parameters

name of type `DOMString` [p.17]

The name of the entity to reference.

Return Value

`EntityReference` [p.65] The new `EntityReference` object.

Exceptions

DOMException [p.20]	<p>INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.</p> <p>NOT_SUPPORTED_ERR: Raised if this document is an HTML document.</p>
------------------------	--

createProcessingInstruction

Creates a `ProcessingInstruction` [p.66] node given the specified name and data strings.

Parameters

target of type `DOMString` [p.17]

The target part of the processing instruction.

data of type `DOMString`

The data for the node.

Return Value

ProcessingInstruction [p.66]	The new ProcessingInstruction object.
---------------------------------	---------------------------------------

Exceptions

DOMException [p.20]	INVALID_CHARACTER_ERR: Raised if the specified target contains an illegal character.
	NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

createTextNode

Creates a Text [p.60] node given the specified string.

Parameters

data of type DOMString [p.17]
The data for the node.

Return Value

Text [p.60] The new Text object.

No Exceptions

getElementById introduced in DOM Level 2

Returns the Element [p.52] whose ID is given by elementId. If no such element exists, returns null. Behavior is not defined if more than one element has this ID.

Note: The DOM implementation must have information that says which attributes are of type ID. Attributes with the name "ID" are not of type ID unless so defined. Implementations that do not know whether attributes are of type ID or not are expected to return null.

Parameters

elementId of type DOMString [p.17]
The unique id value for an element.

Return Value

Element [p.52] The matching element.

No Exceptions

getElementsByTagName

Returns a NodeList [p.43] of all the Elements [p.52] with a given tag name in the order in which they are encountered in a preorder traversal of the Document tree.

Parameters

tagname of type DOMString [p.17]
The name of the tag to match on. The special value "*" matches all tags.

Return Value

NodeList [p.43]	A new NodeList object containing all the matched Elements [p.52] .
--------------------	--

No Exceptions

getElementsByTagName introduced in **DOM Level 2**

Returns a NodeList [p.43] of all the Elements [p.52] with a given *local name* [p.98] and namespace URI in the order in which they are encountered in a preorder traversal of the Document tree.

Parameters

namespaceURI of type DOMString [p.17]

The *namespace URI* [p.99] of the elements to match on. The special value "*" matches all namespaces.

localName of type DOMString

The *local name* [p.98] of the elements to match on. The special value "*" matches all local names.

Return Value

NodeList [p.43]	A new NodeList object containing all the matched Elements [p.52] .
--------------------	--

No Exceptions

importNode introduced in **DOM Level 2**

Imports a node from another document to this document. The returned node has no parent; (parentNode is null). The source node is not altered or removed from the original document; this method creates a new copy of the source node.

For all nodes, importing a node creates a node object owned by the importing document, with attribute values identical to the source node's nodeName and nodeType, plus the attributes related to namespaces (prefix, localName, and namespaceURI). As in the cloneNode operation on a Node [p.34] , the source node is not altered.

Additional information is copied as appropriate to the nodeType, attempting to mirror the behavior expected if a fragment of XML or HTML source was copied from one document to another, recognizing that the two documents may have different DTDs in the XML case. The following list describes the specifics for each type of node.

ATTRIBUTE_NODE

The ownerElement attribute is set to null and the specified flag is set to true on the generated Attr [p.51] . The *descendants* [p.97] of the source Attr are recursively imported and the resulting nodes reassembled to form the corresponding subtree.

Note that the deep parameter has no effect on Attr [p.51] nodes; they always carry their children with them when imported.

DOCUMENT_FRAGMENT_NODE

If the deep option was set to true, the *descendants* [p.97] of the source element are recursively imported and the resulting nodes reassembled to form the corresponding subtree. Otherwise, this simply generates an empty DocumentFragment [p.24] .

DOCUMENT_NODE

Document nodes cannot be imported.

DOCUMENT_TYPE_NODE

DocumentType [p.62] nodes cannot be imported.

ELEMENT_NODE

Specified attribute nodes of the source element are imported, and the generated Attr [p.51] nodes are attached to the generated Element [p.52]. Default attributes are *not* copied, though if the document being imported into defines default attributes for this element name, those are assigned. If the importNode deep parameter was set to true, the *descendants* [p.97] of the source element are recursively imported and the resulting nodes reassembled to form the corresponding subtree.

ENTITY_NODE

Entity [p.64] nodes can be imported, however in the current release of the DOM the DocumentType [p.62] is readonly. Ability to add these imported nodes to a DocumentType will be considered for addition to a future release of the DOM. On import, the publicId, systemId, and notationName attributes are copied. If a deep import is requested, the *descendants* [p.97] of the the source Entity [p.64] are recursively imported and the resulting nodes reassembled to form the corresponding subtree.

ENTITY_REFERENCE_NODE

Only the EntityReference [p.65] itself is copied, even if a deep import is requested, since the source and destination documents might have defined the entity differently. If the document being imported into provides a definition for this entity name, its value is assigned.

NOTATION_NODE

Notation [p.64] nodes can be imported, however in the current release of the DOM the DocumentType [p.62] is readonly. Ability to add these imported nodes to a DocumentType will be considered for addition to a future release of the DOM. On import, the publicId and systemId attributes are copied. Note that the deep parameter has no effect on Notation [p.64] nodes since they never have any children.

PROCESSING_INSTRUCTION_NODE

The imported node copies its target and data values from those of the source node.

TEXT_NODE, CDATA_SECTION_NODE, COMMENT_NODE

These three types of nodes inheriting from CharacterData [p.47] copy their data and length attributes from those of the source node.

Parameters

importedNode of type Node [p.34]

The node to import.

deep of type boolean

If true, recursively import the subtree under the specified node; if false, import only the node itself, as explained above. This has no effect on Attr [p.51], EntityReference [p.65], and Notation [p.64] nodes.

Return Value

Node [p.34] The imported node that belongs to this Document.

Exceptions

DOMException [p.20]	NOT_SUPPORTED_ERR: Raised if the type of node being imported is not supported.
------------------------	--

Interface *Node*

The Node interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree. While all objects implementing the Node interface expose methods for dealing with children, not all objects implementing the Node interface may have children. For example, Text [p.60] nodes may not have children, and adding children to such nodes results in a DOMException [p.20] being raised.

The attributes nodeName, nodeValue and attributes are included as a mechanism to get at node information without casting down to the specific derived interface. In cases where there is no obvious mapping of these attributes for a specific nodeType (e.g., nodeValue for an Element [p.52] or attributes for a Comment [p.61]), this returns null. Note that the specialized interfaces may contain additional and more convenient mechanisms to get and set the relevant information.

IDL Definition

```
interface Node {

    // NodeType
    const unsigned short      ELEMENT_NODE           = 1;
    const unsigned short      ATTRIBUTE_NODE         = 2;
    const unsigned short      TEXT_NODE              = 3;
    const unsigned short      CDATA_SECTION_NODE     = 4;
    const unsigned short      ENTITY_REFERENCE_NODE  = 5;
    const unsigned short      ENTITY_NODE            = 6;
    const unsigned short      PROCESSING_INSTRUCTION_NODE = 7;
    const unsigned short      COMMENT_NODE           = 8;
    const unsigned short      DOCUMENT_NODE          = 9;
    const unsigned short      DOCUMENT_TYPE_NODE     = 10;
    const unsigned short      DOCUMENT_FRAGMENT_NODE = 11;
    const unsigned short      NOTATION_NODE          = 12;

    readonly attribute DOMString      nodeName;
    attribute DOMString               nodeValue;
    // raises(DOMException) on setting
    // raises(DOMException) on retrieval

    readonly attribute unsigned short  nodeType;
    readonly attribute Node            parentNode;
    readonly attribute NodeList        childNodes;
    readonly attribute Node            firstChild;
    readonly attribute Node            lastChild;
    readonly attribute Node            previousSibling;
```

```

readonly attribute Node          nextSibling;
readonly attribute NamedNodeMap attributes;
// Modified in DOM Level 2:
readonly attribute Document      ownerDocument;
Node          insertBefore(in Node newChild,
                           in Node refChild)
                           raises(DOMException);
Node          replaceChild(in Node newChild,
                           in Node oldChild)
                           raises(DOMException);
Node          removeChild(in Node oldChild)
                           raises(DOMException);
Node          appendChild(in Node newChild)
                           raises(DOMException);

boolean       hasChildNodes();
Node          cloneNode(in boolean deep);
// Modified in DOM Level 2:
void          normalize();
// Introduced in DOM Level 2:
boolean       isSupported(in DOMString feature,
                           in DOMString version);

// Introduced in DOM Level 2:
readonly attribute DOMString      namespaceURI;
// Introduced in DOM Level 2:
attribute DOMString      prefix;
                           // raises(DOMException) on setting

// Introduced in DOM Level 2:
readonly attribute DOMString      localName;
// Introduced in DOM Level 2:
boolean       hasAttributes();
};

```

Definition group *NodeType*

An integer indicating which type of node this is.

Note: Numeric codes up to 200 are reserved to W3C for possible future use.

Defined Constants

```

ATTRIBUTE_NODE
    The node is an Attr [p.51] .
CDATA_SECTION_NODE
    The node is a CDATASection [p.62] .
COMMENT_NODE
    The node is a Comment [p.61] .
DOCUMENT_FRAGMENT_NODE
    The node is a DocumentFragment [p.24] .
DOCUMENT_NODE
    The node is a Document [p.25] .
DOCUMENT_TYPE_NODE
    The node is a DocumentType [p.62] .

```

ELEMENT_NODE

The node is an Element [p.52] .

ENTITY_NODE

The node is an Entity [p.64] .

ENTITY_REFERENCE_NODE

The node is an EntityReference [p.65] .

NOTATION_NODE

The node is a Notation [p.64] .

PROCESSING_INSTRUCTION_NODE

The node is a ProcessingInstruction [p.66] .

TEXT_NODE

The node is a Text [p.60] node.

The values of nodeName, nodeValue, and attributes vary according to the node type as follows:

Interface	nodeName	nodeValue	attributes
Attr	name of attribute	value of attribute	null
CDATASection	#cdata-section	content of the CDATA Section	null
Comment	#comment	content of the comment	null
Document	#document	null	null
DocumentFragment	#document-fragment	null	null
DocumentType	document type name	null	null
Element	tag name	null	NamedNodeMap
Entity	entity name	null	null
EntityReference	name of entity referenced	null	null
Notation	notation name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Text	#text	content of the text node	null

Attributes

attributes of type NamedNodeMap [p.44] , readonly

A NamedNodeMap [p.44] containing the attributes of this node (if it is an Element [p.52]) or null otherwise.

`childNodes` of type `NodeList` [p.43] , readonly

A `NodeList` [p.43] that contains all children of this node. If there are no children, this is a `NodeList` containing no nodes.

`firstChild` of type `Node` [p.34] , readonly

The first child of this node. If there is no such node, this returns `null`.

`lastChild` of type `Node` [p.34] , readonly

The last child of this node. If there is no such node, this returns `null`.

`localName` of type `DOMString` [p.17] , readonly, introduced in **DOM Level 2**

Returns the local part of the *qualified name* [p.99] of this node.

For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `createElement` from the Document [p.25] interface, this is always `null`.

`namespaceURI` of type `DOMString` [p.17] , readonly, introduced in **DOM Level 2**

The *namespace URI* [p.99] of this node, or `null` if it is unspecified.

This is not a computed value that is the result of a namespace lookup based on an examination of the namespace declarations in scope. It is merely the namespace URI given at creation time.

For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `createElement` from the Document [p.25] interface, this is always `null`.

Note: Per the *Namespaces in XML* Specification [Namespaces] an attribute does not inherit its namespace from the element it is attached to. If an attribute is not explicitly given a namespace, it simply has no namespace.

`nextSibling` of type `Node` [p.34] , readonly

The node immediately following this node. If there is no such node, this returns `null`.

`nodeName` of type `DOMString` [p.17] , readonly

The name of this node, depending on its type; see the table above.

`nodeType` of type `unsigned short`, readonly

A code representing the type of the underlying object, as defined above.

`nodeValue` of type `DOMString` [p.17]

The value of this node, depending on its type; see the table above. When it is defined to be `null`, setting it has no effect.

Exceptions on setting

<code>DOMException</code> [p.20]	<code>NO_MODIFICATION_ALLOWED_ERR</code> : Raised when the node is readonly.
-------------------------------------	--

Exceptions on retrieval

<code>DOMException</code> [p.20]	<code>DOMSTRING_SIZE_ERR</code> : Raised when it would return more characters than fit in a <code>DOMString</code> [p.17] variable on the implementation platform.
-------------------------------------	--

ownerDocument of type Document [p.25] , readonly, modified in **DOM Level 2**

The Document [p.25] object associated with this node. This is also the Document object used to create new nodes. When this node is a Document or a DocumentType [p.62] which is not used with any Document yet, this is null.

parentNode of type Node [p.34] , readonly

The *parent* [p.99] of this node. All nodes, except Attr [p.51] , Document [p.25] , DocumentFragment [p.24] , Entity [p.64] , and Notation [p.64] may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

prefix of type DOMString [p.17] , introduced in **DOM Level 2**

The *namespace prefix* [p.99] of this node, or null if it is unspecified.

Note that setting this attribute, when permitted, changes the nodeName attribute, which holds the *qualified name* [p.99] , as well as the tagName and name attributes of the Element [p.52] and Attr [p.51] interfaces, when applicable.

Note also that changing the prefix of an attribute that is known to have a default value, does not make a new attribute with the default value and the original prefix appear, since the namespaceURI and localName do not change.

For nodes of any type other than ELEMENT_NODE and ATTRIBUTE_NODE and nodes created with a DOM Level 1 method, such as createElement from the Document [p.25] interface, this is always null.

Exceptions on setting

DOMException [p.20]	INVALID_CHARACTER_ERR: Raised if the specified prefix contains an illegal character.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
	NAMESPACE_ERR: Raised if the specified prefix is malformed, if the namespaceURI of this node is null, if the specified prefix is "xml" and the namespaceURI of this node is different from "http://www.w3.org/XML/1998/namespace", if this node is an attribute and the specified prefix is "xmlns" and the namespaceURI of this node is different from "http://www.w3.org/2000/xmlns/", or if this node is an attribute and the qualifiedName of this node is "xmlns" [Namespaces].

previousSibling of type Node [p.34] , readonly

The node immediately preceding this node. If there is no such node, this returns null.

Methods

appendChild

Adds the node newChild to the end of the list of children of this node. If the newChild is already in the tree, it is first removed.

Parameters

`newChild` of type `Node` [p.34]

The node to add.

If it is a `DocumentFragment` [p.24] object, the entire contents of the document fragment are moved into the child list of this node

Return Value

`Node` [p.34] The node added.

Exceptions

`DOMException` [p.20] **HIERARCHY_REQUEST_ERR**: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node's *ancestors* [p.97] .

WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this node.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

`cloneNode`

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent; (`parentNode` is `null`).

Cloning an `Element` [p.52] copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child `Text` [p.60] node. Cloning an `Attribute` directly, as opposed to be cloned as part of an `Element` cloning operation, returns a specified attribute (`specified` is `true`). Cloning any other type of node simply returns a copy of this node.

Note that cloning an immutable subtree results in a mutable copy, but the children of an `EntityReference` [p.65] clone are *readonly* [p.99] . In addition, clones of unspecified `Attr` [p.51] nodes are specified. And, cloning `Document` [p.25] , `DocumentType` [p.62] , `Entity` [p.64] , and `Notation` [p.64] nodes is implementation dependent.

Parameters

`deep` of type `boolean`

If `true`, recursively clone the subtree under the specified node; if `false`, clone only the node itself (and its attributes, if it is an `Element` [p.52]).

Return Value

`Node` [p.34] The duplicate node.

No Exceptions

`hasAttributes` introduced in **DOM Level 2**

Returns whether this node (if it is an element) has any attributes.

Return Value

`boolean` `true` if this node has any attributes, `false` otherwise.

No Parameters

No Exceptions

`hasChildNodes`

Returns whether this node has any children.

Return Value

`boolean` `true` if this node has any children, `false` otherwise.

No Parameters

No Exceptions

`insertBefore`

Inserts the node `newChild` before the existing child node `refChild`. If `refChild` is `null`, insert `newChild` at the end of the list of children.

If `newChild` is a `DocumentFragment` [p.24] object, all of its children are inserted, in the same order, before `refChild`. If the `newChild` is already in the tree, it is first removed.

Parameters

`newChild` of type `Node` [p.34]

The node to insert.

`refChild` of type `Node`

The reference node, i.e., the node before which the new node must be inserted.

Return Value

`Node` [p.34] The node being inserted.

Exceptions

`DOMException` [p.20]

HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to insert is one of this node's *ancestors* [p.97] .

WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this node.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly or if the parent of the node being inserted is readonly.

NOT_FOUND_ERR: Raised if `refChild` is not a child of this node.

`isSupported` introduced in **DOM Level 2**

Tests whether the DOM implementation implements a specific feature and that feature is supported by this node.

Parameters

feature of type `DOMString` [p.17]

The name of the feature to test. This is the same name which can be passed to the method `hasFeature` on `DOMImplementation` [p.22] .

version of type `DOMString`

This is the version number of the feature to test. In Level 2, version 1, this is the string "2.0". If the version is not specified, supporting any version of the feature will cause the method to return `true`.

Return Value

`boolean` Returns `true` if the specified feature is supported on this node, `false` otherwise.

No Exceptions

`normalize` modified in **DOM Level 2**

Puts all `Text` [p.60] nodes in the full depth of the sub-tree underneath this `Node`, including attribute nodes, into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates `Text` nodes, i.e., there are neither adjacent `Text` nodes nor empty `Text` nodes. This can be used to ensure that the DOM view of a document is the same as if it were saved and re-loaded, and is useful when operations (such as `XPointer` [XPointer] lookups) that depend on a particular document tree structure are to be used.

Note: In cases where the document contains `CDATASections` [p.62] , the `normalize` operation alone may not be sufficient, since `XPointers` do not differentiate between `Text` [p.60] nodes and `CDATASection` [p.62] nodes.

No Parameters

No Return Value

No Exceptions`removeChild`

Removes the child node indicated by `oldChild` from the list of children, and returns it.

Parameters

`oldChild` of type `Node` [p.34]

The node being removed.

Return Value

`Node` [p.34] The node removed.

Exceptions

`DOMException`
[p.20]

`NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is readonly.

`NOT_FOUND_ERR`: Raised if `oldChild` is not a child of this node.

`replaceChild`

Replaces the child node `oldChild` with `newChild` in the list of children, and returns the `oldChild` node.

If `newChild` is a `DocumentFragment` [p.24] object, `oldChild` is replaced by all of the `DocumentFragment` children, which are inserted in the same order. If the `newChild` is already in the tree, it is first removed.

Parameters

`newChild` of type `Node` [p.34]

The new node to put in the child list.

`oldChild` of type `Node`

The node being replaced in the list.

Return Value

`Node` [p.34] The node replaced.

Exceptions

DOMException [p.20]	<p>HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the <code>newChild</code> node, or if the node to put in is one of this node's <i>ancestors</i> [p.97] .</p> <p>WRONG_DOCUMENT_ERR: Raised if <code>newChild</code> was created from a different document than the one that created this node.</p> <p>NO_MODIFICATION_ALLOWED_ERR: Raised if this node or the parent of the new node is readonly.</p> <p>NOT_FOUND_ERR: Raised if <code>oldChild</code> is not a child of this node.</p>
------------------------	---

Interface *NodeList*

The `NodeList` interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented. `NodeList` objects in the DOM are *live* [p.16] .

The items in the `NodeList` are accessible via an integral index, starting from 0.

IDL Definition

```
interface NodeList {
    Node          item(in unsigned long index);
    readonly attribute unsigned long    length;
};
```

Attributes

`length` of type `unsigned long`, readonly
The number of nodes in the list. The range of valid child node indices is 0 to `length-1` inclusive.

Methods

`item`
Returns the `index`th item in the collection. If `index` is greater than or equal to the number of nodes in the list, this returns `null`.

Parameters

`index` of type `unsigned long`
Index into the collection.

Return Value

`Node`
[p.34]
The node at the `index`th position in the `NodeList`, or `null` if that is not a valid index.

No Exceptions

Interface *NamedNodeMap*

Objects implementing the `NamedNodeMap` interface are used to represent collections of nodes that can be accessed by name. Note that `NamedNodeMap` does not inherit from `NodeList` [p.43] ; `NamedNodeMaps` are not maintained in any particular order. Objects contained in an object implementing `NamedNodeMap` may also be accessed by an ordinal index, but this is simply to allow convenient enumeration of the contents of a `NamedNodeMap`, and does not imply that the DOM specifies an order to these Nodes.

`NamedNodeMap` objects in the DOM are *live* [p.16] .

IDL Definition

```
interface NamedNodeMap {
    Node                getNamedItem(in DOMString name);
    Node                setNamedItem(in Node arg)
                        raises(DOMException);
    Node                removeNamedItem(in DOMString name)
                        raises(DOMException);
    Node                item(in unsigned long index);
    readonly attribute unsigned long    length;
    // Introduced in DOM Level 2:
    Node                getNamedItemNS(in DOMString namespaceURI,
                                      in DOMString localName);
    // Introduced in DOM Level 2:
    Node                setNamedItemNS(in Node arg)
                        raises(DOMException);
    // Introduced in DOM Level 2:
    Node                removeNamedItemNS(in DOMString namespaceURI,
                                         in DOMString localName)
                        raises(DOMException);
};
```

Attributes

`length` of type `unsigned long`, `readonly`
 The number of nodes in this map. The range of valid child node indices is 0 to `length-1` inclusive.

Methods

`getNamedItem`
 Retrieves a node specified by name.
Parameters
`name` of type `DOMString` [p.17]
 The `nodeName` of a node to retrieve.

Return Value

`Node`
 [p.34] A `Node` (of any type) with the specified `nodeName`, or `null` if it does not identify any node in this map.

No Exceptions

`getNamedItemNS` introduced in **DOM Level 2**

Retrieves a node specified by local name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters

`namespaceURI` of type `DOMString` [p.17]

The *namespace URI* [p.99] of the node to retrieve.

`localName` of type `DOMString`

The *local name* [p.98] of the node to retrieve.

Return Value

Node [p.34]	A Node (of any type) with the specified local name and namespace URI, or <code>null</code> if they do not identify any node in this map.
----------------	--

No Exceptions

`item`

Returns the `index`th item in the map. If `index` is greater than or equal to the number of nodes in this map, this returns `null`.

Parameters

`index` of type `unsigned long`

Index into this map.

Return Value

Node [p.34]	The node at the <code>index</code> th position in the map, or <code>null</code> if that is not a valid index.
----------------	---

No Exceptions

`removeNamedItem`

Removes a node specified by name. When this map contains the attributes attached to an element, if the removed attribute is known to have a default value, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

Parameters

`name` of type `DOMString` [p.17]

The `nodeName` of the node to remove.

Return Value

Node [p.34]	The node removed from this map if a node with such a name exists.
-------------	---

Exceptions

DOMException [p.20]	NOT_FOUND_ERR: Raised if there is no node named <i>name</i> in this map.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

`removeNamedItemNS` introduced in **DOM Level 2**

Removes a node specified by local name and namespace URI. A removed attribute may be known to have a default value when this map contains the attributes attached to an element, as returned by the `attributes` attribute of the `Node` [p.34] interface. If so, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

HTML-only DOM implementations do not need to implement this method.

Parameters

`namespaceURI` of type `DOMString` [p.17]

The *namespace URI* [p.99] of the node to remove.

`localName` of type `DOMString`

The *local name* [p.98] of the node to remove.

Return Value

Node [p.34]	The node removed from this map if a node with such a local name and namespace URI exists.
----------------	---

Exceptions

DOMException [p.20]	NOT_FOUND_ERR: Raised if there is no node with the specified <code>namespaceURI</code> and <code>localName</code> in this map.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

`setNamedItem`

Adds a node using its `nodeName` attribute. If a node with that name is already present in this map, it is replaced by the new one.

As the `nodeName` attribute is used to derive the name which the node must be stored under, multiple nodes of certain types (those that have a "special" string value) cannot be stored as the names would clash. This is seen as preferable to allowing nodes to be aliased.

Parameters

`arg` of type `Node` [p.34]

A node to store in this map. The node will later be accessible using the value of its `nodeName` attribute.

Return Value

Node [p.34]	If the new Node replaces an existing node the replaced Node is returned, otherwise null is returned.
----------------	--

Exceptions

DOMException [p.20]	<p>WRONG_DOCUMENT_ERR: Raised if <code>arg</code> was created from a different document than the one that created this map.</p> <p>NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.</p> <p>INUSE_ATTRIBUTE_ERR: Raised if <code>arg</code> is an <code>Attr</code> [p.51] that is already an attribute of another <code>Element</code> [p.52] object. The DOM user must explicitly clone <code>Attr</code> nodes to re-use them in other elements.</p>
------------------------	--

`setNamedItemNS` introduced in **DOM Level 2**

Adds a node using its `namespaceURI` and `localName`. If a node with that namespace URI and that local name is already present in this map, it is replaced by the new one. HTML-only DOM implementations do not need to implement this method.

Parameters

`arg` of type Node [p.34]

A node to store in this map. The node will later be accessible using the value of its `namespaceURI` and `localName` attributes.

Return Value

Node [p.34]	If the new Node replaces an existing node the replaced Node is returned, otherwise null is returned.
----------------	--

Exceptions

DOMException [p.20]	<p>WRONG_DOCUMENT_ERR: Raised if <code>arg</code> was created from a different document than the one that created this map.</p> <p>NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.</p> <p>INUSE_ATTRIBUTE_ERR: Raised if <code>arg</code> is an <code>Attr</code> [p.51] that is already an attribute of another <code>Element</code> [p.52] object. The DOM user must explicitly clone <code>Attr</code> nodes to re-use them in other elements.</p>
------------------------	--

Interface *CharacterData*

The `CharacterData` interface extends `Node` with a set of attributes and methods for accessing character data in the DOM. For clarity this set is defined here rather than on each object that uses these attributes and methods. No DOM objects correspond directly to `CharacterData`, though `Text` [p.60] and others do inherit the interface from it. All `offsets` in this interface start from 0.

As explained in the `DOMString` [p.17] interface, text strings in the DOM are represented in UTF-16, i.e. as a sequence of 16-bit units. In the following, the term *16-bit units* [p.97] is used whenever necessary to indicate that indexing on `CharacterData` is done in 16-bit units.

IDL Definition

```
interface CharacterData : Node {
    attribute DOMString      data;
                                // raises(DOMException) on setting
                                // raises(DOMException) on retrieval

    readonly attribute unsigned long    length;
    DOMString      substringData(in unsigned long offset,
                                in unsigned long count)
                                raises(DOMException);
    void            appendData(in DOMString arg)
                                raises(DOMException);
    void            insertData(in unsigned long offset,
                                in DOMString arg)
                                raises(DOMException);
    void            deleteData(in unsigned long offset,
                                in unsigned long count)
                                raises(DOMException);
    void            replaceData(in unsigned long offset,
                                in unsigned long count,
                                in DOMString arg)
                                raises(DOMException);
};
```

Attributes

`data` of type `DOMString` [p.17]

The character data of the node that implements this interface. The DOM implementation may not put arbitrary limits on the amount of data that may be stored in a `CharacterData` node. However, implementation limits may mean that the entirety of a node's data may not fit into a single `DOMString` [p.17]. In such cases, the user may call `substringData` to retrieve the data in appropriately sized pieces.

Exceptions on setting

<code>DOMException</code> [p.20]	<code>NO_MODIFICATION_ALLOWED_ERR</code> : Raised when the node is readonly.
-------------------------------------	--

Exceptions on retrieval

DOMException [p.20]	DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a DOMString [p.17] variable on the implementation platform.
------------------------	---

length of type unsigned long, readonly

The number of *16-bit units* [p.97] that are available through data and the substringData method below. This may have the value zero, i.e., CharacterData nodes may be empty.

Methods

appendData

Append the string to the end of the character data of the node. Upon success, data provides access to the concatenation of data and the DOMString [p.17] specified.

Parameters

arg of type DOMString [p.17]

The DOMString to append.

Exceptions

DOMException [p.20]	NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
------------------------	---

No Return Value

deleteData

Remove a range of *16-bit units* [p.97] from the node. Upon success, data and length reflect the change.

Parameters

offset of type unsigned long

The offset from which to start removing.

count of type unsigned long

The number of 16-bit units to delete. If the sum of offset and count exceeds length then all 16-bit units from offset to the end of the data are deleted.

Exceptions

DOMException [p.20]	INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.
------------------------	---

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

insertData

Insert a string at the specified *16-bit unit* [p.97] offset.

Parameters

offset of type unsigned long

The character offset at which to insert.

arg of type DOMString [p.17]

The DOMString to insert.

Exceptions

DOMException
[p.20]

INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

replaceData

Replace the characters starting at the specified *16-bit unit* [p.97] offset with the specified string.

Parameters

offset of type unsigned long

The offset from which to start replacing.

count of type unsigned long

The number of 16-bit units to replace. If the sum of offset and count exceeds length, then all 16-bit units to the end of the data are replaced; (i.e., the effect is the same as a remove method call with the same range, followed by an append method invocation).

arg of type DOMString [p.17]

The DOMString with which the range must be replaced.

Exceptions

DOMException
[p.20]

INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

substringData

Extracts a range of data from the node.

Parameters

offset of type unsigned long

Start offset of substring to extract.

count of type unsigned long

The number of 16-bit units to extract.

Return Value

DOMString [p.17]	The specified substring. If the sum of <code>offset</code> and <code>count</code> exceeds the <code>length</code> , then all 16-bit units to the end of the data are returned.
---------------------	--

Exceptions

DOMException [p.20]	<p>INDEX_SIZE_ERR: Raised if the specified <code>offset</code> is negative or greater than the number of 16-bit units in <code>data</code>, or if the specified <code>count</code> is negative.</p> <p>DOMSTRING_SIZE_ERR: Raised if the specified range of text does not fit into a DOMString [p.17] .</p>
------------------------	---

Interface Attr

The `Attr` interface represents an attribute in an `Element` [p.52] object. Typically the allowable values for the attribute are defined in a document type definition.

`Attr` objects inherit the `Node` [p.34] interface, but since they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree. Thus, the `Node` attributes `parentNode`, `previousSibling`, and `nextSibling` have a null value for `Attr` objects. The DOM takes the view that attributes are properties of elements rather than having a separate identity from the elements they are associated with; this should make it more efficient to implement such features as default attributes associated with all elements of a given type. Furthermore, `Attr` nodes may not be immediate children of a `DocumentFragment` [p.24] . However, they can be associated with `Element` [p.52] nodes contained within a `DocumentFragment`. In short, users and implementors of the DOM need to be aware that `Attr` nodes have some things in common with other objects inheriting the `Node` interface, but they also are quite distinct.

The attribute's effective value is determined as follows: if this attribute has been explicitly assigned any value, that value is the attribute's effective value; otherwise, if there is a declaration for this attribute, and that declaration includes a default value, then that default value is the attribute's effective value; otherwise, the attribute does not exist on this element in the structure model until it has been explicitly added. Note that the `nodeValue` attribute on the `Attr` instance can also be used to retrieve the string version of the attribute's value(s).

In XML, where the value of an attribute can contain entity references, the child nodes of the `Attr` node may be either `Text` [p.60] or `EntityReference` [p.65] nodes (when these are in use; see the description of `EntityReference` for discussion). Because the DOM Core is not aware of attribute types, it treats all attribute values as simple strings, even if the DTD or schema declares them as having *tokenized* [p.99] types.

IDL Definition

```

interface Attr : Node {
    readonly attribute DOMString      name;
    readonly attribute boolean        specified;
    attribute DOMString               value;
                                     // raises(DOMException) on setting

    // Introduced in DOM Level 2:
    readonly attribute Element        ownerElement;
};

```

Attributes

name of type DOMString [p.17], readonly

Returns the name of this attribute.

ownerElement of type Element [p.52], readonly, introduced in **DOM Level 2**

The Element [p.52] node this attribute is attached to or null if this attribute is not in use.

specified of type boolean, readonly

If this attribute was explicitly given a value in the original document, this is `true`; otherwise, it is `false`. Note that the implementation is in charge of this attribute, not the user. If the user changes the value of the attribute (even if it ends up having the same value as the default value) then the `specified` flag is automatically flipped to `true`. To re-specify the attribute as the default value from the DTD, the user must delete the attribute. The implementation will then make a new attribute available with `specified` set to `false` and the default value (if one exists).

In summary:

- If the attribute has an assigned value in the document then `specified` is `true`, and the value is the assigned value.
- If the attribute has no assigned value in the document and has a default value in the DTD, then `specified` is `false`, and the value is the default value in the DTD.
- If the attribute has no assigned value in the document and has a value of `#IMPLIED` in the DTD, then the attribute does not appear in the structure model of the document.
- If the `ownerElement` attribute is null (i.e. because it was just created or was set to null by the various removal and cloning operations) `specified` is `true`.

value of type DOMString [p.17]

On retrieval, the value of the attribute is returned as a string. Character and general entity references are replaced with their values. See also the method `getAttribute` on the Element [p.52] interface.

On setting, this creates a Text [p.60] node with the unparsed contents of the string. I.e. any characters that an XML processor would recognize as markup are instead treated as literal text. See also the method `setAttribute` on the Element [p.52] interface.

Exceptions on setting

DOMException [p.20]	NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.
------------------------	--

Interface *Element*

The `Element` interface represents an *element* [p.98] in an HTML or XML document. Elements may have attributes associated with them; since the `Element` interface inherits from `Node` [p.34], the generic `Node` interface attribute `attributes` may be used to retrieve the set of all attributes for an element. There are methods on the `Element` interface to retrieve either an `Attr` [p.51] object by name or an attribute value by name. In XML, where an attribute value may contain entity references, an `Attr` object should be retrieved to examine the possibly fairly complex sub-tree representing the attribute value. On the other hand, in HTML, where all attributes have simple string values, methods to directly access an attribute value can safely be used as a *convenience* [p.97].

Note: In DOM Level 2, the method `normalize` is inherited from the `Node` [p.34] interface where it was moved.

IDL Definition

```
interface Element : Node {
    readonly attribute DOMString      tagName;
    DOMString      getAttribute(in DOMString name);
    void           setAttribute(in DOMString name,
                               in DOMString value)
                               raises(DOMException);
    void           removeAttribute(in DOMString name)
                               raises(DOMException);
    Attr           getAttributeNode(in DOMString name);
    Attr           setAttributeNode(in Attr newAttr)
                               raises(DOMException);
    Attr           removeAttributeNode(in Attr oldAttr)
                               raises(DOMException);
    NodeList       getElementsByTagName(in DOMString name);
    // Introduced in DOM Level 2:
    DOMString      getAttributeNS(in DOMString namespaceURI,
                                  in DOMString localName);
    // Introduced in DOM Level 2:
    void           setAttributeNS(in DOMString namespaceURI,
                                  in DOMString qualifiedName,
                                  in DOMString value)
                                  raises(DOMException);
    // Introduced in DOM Level 2:
    void           removeAttributeNS(in DOMString namespaceURI,
                                     in DOMString localName)
                                     raises(DOMException);
    // Introduced in DOM Level 2:
    Attr           getAttributeNodeNS(in DOMString namespaceURI,
                                       in DOMString localName);
    // Introduced in DOM Level 2:
    Attr           setAttributeNodeNS(in Attr newAttr)
                                       raises(DOMException);
    // Introduced in DOM Level 2:
    NodeList       getElementsByTagNameNS(in DOMString namespaceURI,
                                          in DOMString localName);
    // Introduced in DOM Level 2:
    boolean        hasAttribute(in DOMString name);
```

```
// Introduced in DOM Level 2:
boolean      hasAttributeNS(in DOMString namespaceURI,
                           in DOMString localName);
};
```

Attributes

tagName of type DOMString [p.17] , readonly
The name of the element. For example, in:

```
<elementExample id="demo">
    ...
</elementExample> ,
```

tagName has the value "elementExample". Note that this is case-preserving in XML, as are all of the operations of the DOM. The HTML DOM returns the tagName of an HTML element in the canonical uppercase form, regardless of the case in the source HTML document.

Methods

getAttribute

Retrieves an attribute value by name.

Parameters

name of type DOMString [p.17]

The name of the attribute to retrieve.

Return Value

DOMString [p.17]	The Attr [p.51] value as a string, or the empty string if that attribute does not have a specified or default value.
---------------------	--

No Exceptions

getAttributeNS introduced in **DOM Level 2**

Retrieves an attribute value by local name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString [p.17]

The *namespace URI* [p.99] of the attribute to retrieve.

localName of type DOMString

The *local name* [p.98] of the attribute to retrieve.

Return Value

DOMString [p.17]	The Attr [p.51] value as a string, or the empty string if that attribute does not have a specified or default value.
---------------------	--

No Exceptions

getAttributeNode

Retrieves an attribute node by name.

To retrieve an attribute node by qualified name and namespace URI, use the getAttributeNodeNS method.

Parameters

name of type DOMString [p.17]

The name (nodeName) of the attribute to retrieve.

Return Value

Attr [p.51] The Attr node with the specified name (nodeName) or null if there is no such attribute.

No Exceptions

getAttributeNodeNS introduced in **DOM Level 2**

Retrieves an Attr [p.51] node by local name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString [p.17]

The *namespace URI* [p.99] of the attribute to retrieve.

localName of type DOMString

The *local name* [p.98] of the attribute to retrieve.

Return Value

Attr [p.51] The Attr node with the specified attribute local name and namespace URI or null if there is no such attribute.

No Exceptions

getElementsByTagName

Returns a NodeList [p.43] of all *descendant* [p.97] Elements with a given tag name, in the order in which they are encountered in a preorder traversal of this Element tree.

Parameters

name of type DOMString [p.17]

The name of the tag to match on. The special value "*" matches all tags.

Return Value

NodeList [p.43] A list of matching Element nodes.

No Exceptions

getElementsByTagNameNS introduced in **DOM Level 2**

Returns a NodeList [p.43] of all the *descendant* [p.97] Elements with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of this Element tree.

HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString [p.17]

The *namespace URI* [p.99] of the elements to match on. The special value "*" matches all namespaces.

localName of type DOMString

The *local name* [p.98] of the elements to match on. The special value "*" matches all local names.

Return Value

NodeList [p.43]	A new NodeList object containing all the matched Elements.
--------------------	--

No Exceptions

hasAttribute introduced in **DOM Level 2**

Returns `true` when an attribute with a given name is specified on this element or has a default value, `false` otherwise.

Parameters

name of type DOMString [p.17]

The name of the attribute to look for.

Return Value

boolean	<code>true</code> if an attribute with the given name is specified on this element or has a default value, <code>false</code> otherwise.
---------	--

No Exceptions

hasAttributeNS introduced in **DOM Level 2**

Returns `true` when an attribute with a given local name and namespace URI is specified on this element or has a default value, `false` otherwise. HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString [p.17]

The *namespace URI* [p.99] of the attribute to look for.

localName of type DOMString

The *local name* [p.98] of the attribute to look for.

Return Value

boolean	<code>true</code> if an attribute with the given local name and namespace URI is specified or has a default value on this element, <code>false</code> otherwise.
---------	--

No Exceptions

removeAttribute

Removes an attribute by name. If the removed attribute is known to have a default value, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

To remove an attribute by local name and namespace URI, use the `removeAttributeNS` method.

Parameters

name of type DOMString [p.17]

The name of the attribute to remove.

Exceptions

DOMException
[p.20]

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

removeAttributeNS introduced in **DOM Level 2**

Removes an attribute by local name and namespace URI. If the removed attribute has a default value it is immediately replaced. The replacing attribute has the same namespace URI and local name, as well as the original prefix.

HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type DOMString [p.17]

The *namespace URI* [p.99] of the attribute to remove.

localName of type DOMString

The *local name* [p.98] of the attribute to remove.

Exceptions

DOMException
[p.20]

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

removeAttributeNode

Removes the specified attribute node. If the removed Attr [p.51] has a default value it is immediately replaced. The replacing attribute has the same namespace URI and local name, as well as the original prefix, when applicable.

Parameters

oldAttr of type Attr [p.51]

The Attr node to remove from the attribute list.

Return Value

Attr [p.51] The Attr node that was removed.

Exceptions

DOMException
[p.20]

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

NOT_FOUND_ERR: Raised if oldAttr is not an attribute of the element.

setAttribute

Adds a new attribute. If an attribute with that name is already present in the element, its value is changed to be that of the value parameter. This value is a simple string; it is not parsed as it is being set. So any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately escaped by the implementation when it is written out. In order to assign an attribute value that contains entity references, the user must create an `Attr` [p.51] node plus any `Text` [p.60] and `EntityReference` [p.65] nodes, build the appropriate subtree, and use `setAttributeNode` to assign it as the value of an attribute.

To set an attribute with a qualified name and namespace URI, use the `setAttributeNS` method.

Parameters

name of type `DOMString` [p.17]

The name of the attribute to create or alter.

value of type `DOMString`

Value to set in string form.

Exceptions

<code>DOMException</code> [p.20]	<code>INVALID_CHARACTER_ERR</code> : Raised if the specified name contains an illegal character.
-------------------------------------	--

<code>NO_MODIFICATION_ALLOWED_ERR</code> : Raised if this node is readonly.

No Return Value**setAttributeNS introduced in DOM Level 2**

Adds a new attribute. If an attribute with the same local name and namespace URI is already present on the element, its prefix is changed to be the prefix part of the `qualifiedName`, and its value is changed to be the value parameter. This value is a simple string; it is not parsed as it is being set. So any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately escaped by the implementation when it is written out. In order to assign an attribute value that contains entity references, the user must create an `Attr` [p.51] node plus any `Text` [p.60] and `EntityReference` [p.65] nodes, build the appropriate subtree, and use `setAttributeNodeNS` or `setAttributeNode` to assign it as the value of an attribute.

HTML-only DOM implementations do not need to implement this method.

Parameters

namespaceURI of type `DOMString` [p.17]

The *namespace URI* [p.99] of the attribute to create or alter.

qualifiedName of type `DOMString`

The *qualified name* [p.99] of the attribute to create or alter.

value of type `DOMString`

The value to set in string form.

Exceptions

`DOMException` [p.20] `INVALID_CHARACTER_ERR`: Raised if the specified qualified name contains an illegal character.

`NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is readonly.

`NAMESPACE_ERR`: Raised if the `qualifiedName` is malformed, if the `qualifiedName` has a prefix and the `namespaceURI` is null, if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from "http://www.w3.org/XML/1998/namespace", or if the `qualifiedName` is "xmlns" and the `namespaceURI` is different from "http://www.w3.org/2000/xmlns/".

No Return Value

`setAttributeNode`

Adds a new attribute node. If an attribute with that name (`nodeName`) is already present in the element, it is replaced by the new one.

To add a new attribute node with a qualified name and namespace URI, use the `setAttributeNodeNS` method.

Parameters

`newAttr` of type `Attr` [p.51]

The `Attr` node to add to the attribute list.

Return Value

`Attr` [p.51] If the `newAttr` attribute replaces an existing attribute, the replaced `Attr` node is returned, otherwise `null` is returned.

Exceptions

`DOMException` [p.20] `WRONG_DOCUMENT_ERR`: Raised if `newAttr` was created from a different document than the one that created the element.

`NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is readonly.

`INUSE_ATTRIBUTE_ERR`: Raised if `newAttr` is already an attribute of another `Element` object. The DOM user must explicitly clone `Attr` [p.51] nodes to re-use them in other elements.

`setAttributeNodeNS` introduced in **DOM Level 2**

Adds a new attribute. If an attribute with that local name and that namespace URI is already present in the element, it is replaced by the new one.

HTML-only DOM implementations do not need to implement this method.

Parameters

`newAttr` of type `Attr` [p.51]

The `Attr` node to add to the attribute list.

Return Value

`Attr` [p.51] If the `newAttr` attribute replaces an existing attribute with the same *local name* [p.98] and *namespace URI* [p.99], the replaced `Attr` node is returned, otherwise `null` is returned.

Exceptions

`DOMException` [p.20] `WRONG_DOCUMENT_ERR`: Raised if `newAttr` was created from a different document than the one that created the element.

`NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is readonly.

`INUSE_ATTRIBUTE_ERR`: Raised if `newAttr` is already an attribute of another `Element` object. The DOM user must explicitly clone `Attr` [p.51] nodes to re-use them in other elements.

Interface *Text*

The `Text` interface inherits from `CharacterData` [p.47] and represents the textual content (termed *character data* in XML) of an `Element` [p.52] or `Attr` [p.51]. If there is no markup inside an element's content, the text is contained in a single object implementing the `Text` interface that is the only child of the element. If there is markup, it is parsed into the *information items* [p.98] (elements, comments, etc.) and `Text` nodes that form the list of children of the element.

When a document is first made available via the DOM, there is only one `Text` node for each block of text. Users may create adjacent `Text` nodes that represent the contents of a given element without any intervening markup, but should be aware that there is no way to represent the separations between these nodes in XML or HTML, so they will not (in general) persist between DOM editing sessions. The `normalize()` method on `Node` [p.34] merges any such adjacent `Text` objects into a single node for each block of text.

IDL Definition

```
interface Text : CharacterData {
    Text          splitText(in unsigned long offset)
                                   raises(DOMException);
};
```

Methods

`splitText`
Breaks this node into two nodes at the specified `offset`, keeping both in the tree as *siblings* [p.99]. After being split, this node will contain all the content up to the `offset`

point. A new node of the same type, which contains all the content at and after the `offset` point, is returned. If the original node had a parent node, the new node is inserted as the next *sibling* [p.99] of the original node. When the `offset` is equal to the length of this node, the new node has no data.

Parameters

`offset` of type `unsigned long`

The *16-bit unit* [p.97] offset at which to split, starting from 0.

Return Value

Text [p.60] The new node, of the same type as this node.

Exceptions

DOMException [p.20] INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in `data`.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is `readonly`.

Interface *Comment*

This interface inherits from `CharacterData` [p.47] and represents the content of a comment, i.e., all the characters between the starting '`<!--`' and ending '`-->`'. Note that this is the definition of a comment in XML, and, in practice, HTML, although some HTML tools may implement the full SGML comment structure.

IDL Definition

```
interface Comment : CharacterData {
};
```

1.3. Extended Interfaces

The interfaces defined here form part of the DOM Core specification, but objects that expose these interfaces will never be encountered in a DOM implementation that deals only with HTML. As such, HTML-only DOM implementations [DOM Level 2 HTML] do not need to have objects that implement these interfaces.

The interfaces found within this section are not mandatory. A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` [p.22] interface with parameter values "XML" and "2.0" (respectively) to determine whether or not this module is supported by the implementation. In order to fully support this module, an implementation must also support the "Core" feature defined in Fundamental Interfaces [p.20]. Please refer to additional information about Conformance [p.12] in this specification.

Interface *CDATASection*

CDATA sections are used to escape blocks of text containing characters that would otherwise be regarded as markup. The only delimiter that is recognized in a CDATA section is the "]]>" string that ends the CDATA section. CDATA sections cannot be nested. Their primary purpose is for including material such as XML fragments, without needing to escape all the delimiters.

The `DOMString` [p.17] attribute of the `Text` [p.60] node holds the text that is contained by the CDATA section. Note that this *may* contain characters that need to be escaped outside of CDATA sections and that, depending on the character encoding ("charset") chosen for serialization, it may be impossible to write out some characters as part of a CDATA section.

The `CDATASection` interface inherits from the `CharacterData` [p.47] interface through the `Text` [p.60] interface. Adjacent `CDATASection` nodes are not merged by use of the `normalize` method of the `Node` [p.34] interface.

Note: Because no markup is recognized within a `CDATASection`, character numeric references cannot be used as an escape mechanism when serializing. Therefore, action needs to be taken when serializing a `CDATASection` with a character encoding where some of the contained characters cannot be represented. Failure to do so would not produce well-formed XML.

One potential solution in the serialization process is to end the CDATA section before the character, output the character using a character reference or entity reference, and open a new CDATA section for any further characters in the text node. Note, however, that some code conversion libraries at the time of writing do not return an error or exception when a character is missing from the encoding, making the task of ensuring that data is not corrupted on serialization more difficult.

IDL Definition

```
interface CDATASection : Text {
};
```

Interface *DocumentType*

Each `Document` [p.25] has a `doctype` attribute whose value is either `null` or a `DocumentType` object. The `DocumentType` interface in the DOM Core provides an interface to the list of entities that are defined for the document, and little else because the effect of namespaces and the various XML schema efforts on DTD representation are not clearly understood as of this writing.

The DOM Level 2 doesn't support editing `DocumentType` nodes.

IDL Definition

```

interface DocumentType : Node {
    readonly attribute DOMString      name;
    readonly attribute NamedNodeMap   entities;
    readonly attribute NamedNodeMap   notations;
    // Introduced in DOM Level 2:
    readonly attribute DOMString      publicId;
    // Introduced in DOM Level 2:
    readonly attribute DOMString      systemId;
    // Introduced in DOM Level 2:
    readonly attribute DOMString      internalSubset;
};

```

Attributes

`entities` of type `NamedNodeMap` [p.44] , `readonly`

A `NamedNodeMap` [p.44] containing the general entities, both external and internal, declared in the DTD. Parameter entities are not contained. Duplicates are discarded. For example in:

```

<!DOCTYPE ex SYSTEM "ex.dtd" [
    <!ENTITY foo "foo">
    <!ENTITY bar "bar">
    <!ENTITY bar "bar2">
    <!ENTITY % baz "baz">
]>
<ex/>

```

the interface provides access to `foo` and the first declaration of `bar` but not the second declaration of `bar` or `baz`. Every node in this map also implements the `Entity` [p.64] interface.

The DOM Level 2 does not support editing entities, therefore `entities` cannot be altered in any way.

`internalSubset` of type `DOMString` [p.17] , `readonly`, introduced in **DOM Level 2**

The internal subset as a string.

Note: The actual content returned depends on how much information is available to the implementation. This may vary depending on various parameters, including the XML processor used to build the document.

`name` of type `DOMString` [p.17] , `readonly`

The name of DTD; i.e., the name immediately following the `DOCTYPE` keyword.

`notations` of type `NamedNodeMap` [p.44] , `readonly`

A `NamedNodeMap` [p.44] containing the notations declared in the DTD. Duplicates are discarded. Every node in this map also implements the `Notation` [p.64] interface.

The DOM Level 2 does not support editing notations, therefore `notations` cannot be altered in any way.

`publicId` of type `DOMString` [p.17] , `readonly`, introduced in **DOM Level 2**

The public identifier of the external subset.

`systemId` of type `DOMString` [p.17] , `readonly`, introduced in **DOM Level 2**

The system identifier of the external subset.

Interface *Notation*

This interface represents a notation declared in the DTD. A notation either declares, by name, the format of an unparsed entity (see *section 4.7* of the XML 1.0 specification [XML]), or is used for formal declaration of processing instruction targets (see *section 2.6* of the XML 1.0 specification [XML]). The `nodeName` attribute inherited from `Node` [p.34] is set to the declared name of the notation.

The DOM Level 1 does not support editing `Notation` nodes; they are therefore *readonly* [p.99] .

A `Notation` node does not have any parent.

IDL Definition

```
interface Notation : Node {
    readonly attribute DOMString      publicId;
    readonly attribute DOMString      systemId;
};
```

Attributes

`publicId` of type `DOMString` [p.17] , *readonly*

The public identifier of this notation. If the public identifier was not specified, this is `null`.

`systemId` of type `DOMString` [p.17] , *readonly*

The system identifier of this notation. If the system identifier was not specified, this is `null`.

Interface *Entity*

This interface represents an entity, either parsed or unparsed, in an XML document. Note that this models the entity itself *not* the entity declaration. `Entity` declaration modeling has been left for a later Level of the DOM specification.

The `nodeName` attribute that is inherited from `Node` [p.34] contains the name of the entity.

An XML processor may choose to completely expand entities before the structure model is passed to the DOM; in this case there will be no `EntityReference` [p.65] nodes in the document tree.

XML does not mandate that a non-validating XML processor read and process entity declarations made in the external subset or declared in external parameter entities. This means that parsed entities declared in the external subset need not be expanded by some classes of applications, and that the replacement value of the entity may not be available. When the replacement value is available, the corresponding `Entity` node's child list represents the structure of that replacement text. Otherwise, the child list is empty.

The DOM Level 2 does not support editing `Entity` nodes; if a user wants to make changes to the contents of an `Entity`, every related `EntityReference` [p.65] node has to be replaced in the structure model by a clone of the `Entity`'s contents, and then the desired changes must be made to each of those clones instead. `Entity` nodes and all their *descendants* [p.97] are *readonly* [p.99] .

An Entity node does not have any parent.

Note: If the entity contains an unbound *namespace prefix* [p.99] , the namespaceURI of the corresponding node in the Entity node subtree is null. The same is true for EntityReference [p.65] nodes that refer to this entity, when they are created using the createEntityReference method of the Document [p.25] interface. The DOM Level 2 does not support any mechanism to resolve namespace prefixes.

IDL Definition

```
interface Entity : Node {
    readonly attribute DOMString    publicId;
    readonly attribute DOMString    systemId;
    readonly attribute DOMString    notationName;
};
```

Attributes

notationName of type DOMString [p.17] , readonly
 For unparsed entities, the name of the notation for the entity. For parsed entities, this is null.

publicId of type DOMString [p.17] , readonly
 The public identifier associated with the entity, if specified. If the public identifier was not specified, this is null.

systemId of type DOMString [p.17] , readonly
 The system identifier associated with the entity, if specified. If the system identifier was not specified, this is null.

Interface EntityReference

EntityReference objects may be inserted into the structure model when an entity reference is in the source document, or when the user wishes to insert an entity reference. Note that character references and references to predefined entities are considered to be expanded by the HTML or XML processor so that characters are represented by their Unicode equivalent rather than by an entity reference. Moreover, the XML processor may completely expand references to entities while building the structure model, instead of providing EntityReference objects. If it does provide such objects, then for a given EntityReference node, it may be that there is no Entity [p.64] node representing the referenced entity. If such an Entity exists, then the subtree of the EntityReference node is in general a copy of the Entity node subtree. However, this may not be true when an entity contains an unbound *namespace prefix* [p.99] . In such a case, because the namespace prefix resolution depends on where the entity reference is, the *descendants* [p.97] of the EntityReference node may be bound to different *namespace URIs* [p.99] .

As for Entity [p.64] nodes, EntityReference nodes and all their *descendants* [p.97] are *readonly* [p.99] .

IDL Definition

```
interface EntityReference : Node {
};
```

Interface *ProcessingInstruction*

The `ProcessingInstruction` interface represents a "processing instruction", used in XML as a way to keep processor-specific information in the text of the document.

IDL Definition

```
interface ProcessingInstruction : Node {
    readonly attribute DOMString    target;
        attribute DOMString        data;
                                   // raises(DOMException) on setting
};
```

Attributes

data of type `DOMString` [p.17]

The content of this processing instruction. This is from the first non white space character after the target to the character immediately preceding the `?>`.

Exceptions on setting

<code>DOMException</code> [p.20]	<code>NO_MODIFICATION_ALLOWED_ERR</code> : Raised when the node is readonly.
-------------------------------------	--

target of type `DOMString` [p.17] , readonly

The target of this processing instruction. XML defines this as being the first *token* [p.99] following the markup that begins the processing instruction.

Appendix A: Changes

Editors

Arnaud Le Hors, IBM
Philippe Le Hégaré, W3C

A.1: Changes between DOM Level 1 Core and DOM Level 2 Core

OMG IDL

The DOM Level 2 specifications are now using Corba 2.3.1 instead of Corba 2.2.

Type **DOMString** [p.17]

The definition of DOMString [p.17] in IDL is now a `valuetype`.

A.1.1: Changes to DOM Level 1 Core interfaces and exceptions

Interface **Attr** [p.51]

The **Attr** [p.51] interface has one new attribute: `ownerElement`.

Interface **Document** [p.25]

The **Document** [p.25] interface has five new methods: `importNode`, `createElementNS`, `createAttributeNS`, `getElementsByNameNS` and `getElementById`.

Interface **NamedNodeMap** [p.44]

The **NamedNodeMap** [p.44] interface has three new methods: `getNamedItemNS`, `setNamedItemNS`, `removeNamedItemNS`.

Interface **Node** [p.34]

The **Node** [p.34] interface has two new methods: `isSupported` and `hasAttributes`. `normalize`, previously in the **Element** [p.52] interface, has been moved in the **Node** [p.34] interface.

The **Node** [p.34] interface has three new attributes: `namespaceURI`, `prefix` and `localName`. The `ownerDocument` attribute was specified to be `null` when the node is a **Document** [p.25]. It now is also `null` when the node is a **DocumentType** [p.62] which is not used with any **Document** yet.

Interface **DocumentType** [p.62]

The **DocumentType** [p.62] interface has three attributes: `publicId`, `systemId` and `internalSubset`.

Interface **DOMImplementation** [p.22]

The **DOMImplementation** [p.22] interface has two new methods: `createDocumentType` and `createDocument`.

Interface **Element** [p.52]

The **Element** [p.52] interface has eight new methods: `getAttributeNS`, `setAttributeNS`, `removeAttributeNS`, `getAttributeNodeNS`, `setAttributeNodeNS`, `getElementsByNameNS`, `hasAttribute` and `hasAttributeNS`.

The method `normalize` is now inherited from the **Node** [p.34] interface where it was moved.

Exception `DOMException` [p.20]

The `DOMException` [p.20] has five new exception codes: `INVALID_STATE_ERR`, `SYNTAX_ERR`, `INVALID_MODIFICATION_ERR`, `NAMESPACE_ERR` and `INVALID_ACCESS_ERR`.

A.1.2: New features

A.1.2.1: New types

`DOMTimeStamp` [p.18]

The `DOMTimeStamp` [p.18] type was added to the Core module.

Appendix B: Accessing code point boundaries

Mark Davis, IBM

Lauren Wood, SoftQuad Software Inc.

B.1: Introduction

This appendix is an informative, not a normative, part of the Level 2 DOM specification.

Characters are represented in Unicode by numbers called *code points* (also called *scalar values*). These numbers can range from 0 up to $1,114,111 = 10\text{FFFF}_{16}$ (although some of these values are illegal). Each code point can be directly encoded with a 32-bit code unit. This encoding is termed UCS-4 (or UTF-32). The DOM specification, however, uses UTF-16, in which the most frequent characters (which have values less than FFFF_{16}) are represented by a single 16-bit code unit, while characters above FFFF_{16} use a special pair of code units called a *surrogate pair*. For more information, see [Unicode] or the Unicode Web site.

While indexing by code points as opposed to code units is not common in programs, some specifications such as XPath (and therefore XSLT and XPointer) use code point indices. For interfacing with such formats it is recommended that the programming language provide string processing methods for converting code point indices to code unit indices and back. Some languages do not provide these functions natively; for these it is recommended that the native `String` type that is bound to `DOMString` [p.17] be extended to enable this conversion. An example of how such an API might look is supplied below.

Note: Since these methods are supplied as an illustrative example of the type of functionality that is required, the names of the methods, exceptions, and interface may differ from those given here.

B.2: Methods

Interface *StringExtend*

Extensions to a language's native `String` class or interface

IDL Definition

```
interface StringExtend {
    int findOffset16(in int offset32)
                                     raises(StringIndexOutOfBoundsException);
    int findOffset32(in int offset16)
                                     raises(StringIndexOutOfBoundsException);
};
```

Methods

`findOffset16`

Returns the UTF-16 offset that corresponds to a UTF-32 offset. Used for random access.

Note: You can always round-trip from a UTF-32 offset to a UTF-16 offset and back. You can round-trip from a UTF-16 offset to a UTF-32 offset and back if and only if the offset16 is not in the middle of a surrogate pair. Unmatched surrogates count as a single UTF-16 value.

Parameters

offset32 of type int
UTF-32 offset.

Return Value

int UTF-16 offset

Exceptions

`StringIndexOutOfBoundsException` if offset32 is out of bounds.

findOffset32

Returns the UTF-32 offset corresponding to a UTF-16 offset. Used for random access. To find the UTF-32 length of a string, use:

```
len32 = findOffset32(source, source.length());
```

Note: If the UTF-16 offset is into the middle of a surrogate pair, then the UTF-32 offset of the *end* of the pair is returned; that is, the index of the char after the end of the pair. You can always round-trip from a UTF-32 offset to a UTF-16 offset and back. You can round-trip from a UTF-16 offset to a UTF-32 offset and back if and only if the offset16 is not in the middle of a surrogate pair. Unmatched surrogates count as a single UTF-16 value.

Parameters

offset16 of type int
UTF-16 offset

Return Value

int UTF-32 offset

Exceptions

`StringIndexOutOfBoundsException` if offset16 is out of bounds.

Appendix C: IDL Definitions

This appendix contains the complete OMG IDL [OMGIDL] for the Level 2 Document Object Model Core definitions.

The IDL files are also available as:

<http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/idl.zip>

dom.idl:

```
// File: dom.idl

#ifndef _DOM_IDL_
#define _DOM_IDL_

#pragma prefix "w3c.org"
module dom
{

    valuetype DOMString sequence<unsigned short>;

    typedef    unsigned long long DOMTimeStamp;

    interface DocumentType;
    interface Document;
    interface NodeList;
    interface NamedNodeMap;
    interface Element;

    exception DOMException {
        unsigned short    code;
    };
    // ExceptionCode
    const unsigned short    INDEX_SIZE_ERR                = 1;
    const unsigned short    DOMSTRING_SIZE_ERR            = 2;
    const unsigned short    HIERARCHY_REQUEST_ERR         = 3;
    const unsigned short    WRONG_DOCUMENT_ERR            = 4;
    const unsigned short    INVALID_CHARACTER_ERR         = 5;
    const unsigned short    NO_DATA_ALLOWED_ERR           = 6;
    const unsigned short    NO_MODIFICATION_ALLOWED_ERR    = 7;
    const unsigned short    NOT_FOUND_ERR                  = 8;
    const unsigned short    NOT_SUPPORTED_ERR              = 9;
    const unsigned short    INUSE_ATTRIBUTE_ERR            = 10;
    // Introduced in DOM Level 2:
    const unsigned short    INVALID_STATE_ERR              = 11;
    // Introduced in DOM Level 2:
    const unsigned short    SYNTAX_ERR                     = 12;
    // Introduced in DOM Level 2:
    const unsigned short    INVALID_MODIFICATION_ERR       = 13;
    // Introduced in DOM Level 2:
    const unsigned short    NAMESPACE_ERR                  = 14;
    // Introduced in DOM Level 2:
    const unsigned short    INVALID_ACCESS_ERR              = 15;
```


dom.idl:

```
interface DOMImplementation {
    boolean          hasFeature(in DOMString feature,
                                in DOMString version);
    // Introduced in DOM Level 2:
    DocumentType     createDocumentType(in DOMString qualifiedName,
                                         in DOMString publicId,
                                         in DOMString systemId)
                                         raises(DOMException);
    // Introduced in DOM Level 2:
    Document         createDocument(in DOMString namespaceURI,
                                    in DOMString qualifiedName,
                                    in DocumentType doctype)
                                    raises(DOMException);
};

interface Node {

    // NodeType
    const unsigned short    ELEMENT_NODE           = 1;
    const unsigned short    ATTRIBUTE_NODE         = 2;
    const unsigned short    TEXT_NODE              = 3;
    const unsigned short    CDATA_SECTION_NODE     = 4;
    const unsigned short    ENTITY_REFERENCE_NODE  = 5;
    const unsigned short    ENTITY_NODE            = 6;
    const unsigned short    PROCESSING_INSTRUCTION_NODE = 7;
    const unsigned short    COMMENT_NODE           = 8;
    const unsigned short    DOCUMENT_NODE          = 9;
    const unsigned short    DOCUMENT_TYPE_NODE     = 10;
    const unsigned short    DOCUMENT_FRAGMENT_NODE = 11;
    const unsigned short    NOTATION_NODE          = 12;

    readonly attribute DOMString    nodeName;
    attribute DOMString             nodeValue;
    // raises(DOMException) on setting
    // raises(DOMException) on retrieval

    readonly attribute unsigned short    nodeType;
    readonly attribute Node              parentNode;
    readonly attribute NodeList          childNodes;
    readonly attribute Node              firstChild;
    readonly attribute Node              lastChild;
    readonly attribute Node              previousSibling;
    readonly attribute Node              nextSibling;
    readonly attribute NamedNodeMap      attributes;
    // Modified in DOM Level 2:
    readonly attribute Document          ownerDocument;
    Node      insertBefore(in Node newChild,
                           in Node refChild)
                           raises(DOMException);
    Node      replaceChild(in Node newChild,
                           in Node oldChild)
                           raises(DOMException);
    Node      removeChild(in Node oldChild)
                           raises(DOMException);
    Node      appendChild(in Node newChild)
                           raises(DOMException);
};
```

dom.idl:

```
boolean          hasChildNodes();
Node             cloneNode(in boolean deep);
// Modified in DOM Level 2:
void            normalize();
// Introduced in DOM Level 2:
boolean         isSupported(in DOMString feature,
                           in DOMString version);

// Introduced in DOM Level 2:
readonly attribute DOMString      namespaceURI;
// Introduced in DOM Level 2:
attribute DOMString              prefix;
// raises(DOMException) on setting

// Introduced in DOM Level 2:
readonly attribute DOMString      localName;
// Introduced in DOM Level 2:
boolean          hasAttributes();
};

interface NodeList {
    Node          item(in unsigned long index);
    readonly attribute unsigned long    length;
};

interface NamedNodeMap {
    Node          getNamedItem(in DOMString name);
    Node          setNamedItem(in Node arg)
                  raises(DOMException);
    Node          removeNamedItem(in DOMString name)
                  raises(DOMException);
    Node          item(in unsigned long index);
    readonly attribute unsigned long    length;
    // Introduced in DOM Level 2:
    Node          getNamedItemNS(in DOMString namespaceURI,
                                in DOMString localName);
    // Introduced in DOM Level 2:
    Node          setNamedItemNS(in Node arg)
                  raises(DOMException);
    // Introduced in DOM Level 2:
    Node          removeNamedItemNS(in DOMString namespaceURI,
                                   in DOMString localName)
                  raises(DOMException);
};

interface CharacterData : Node {
    attribute DOMString      data;
    // raises(DOMException) on setting
    // raises(DOMException) on retrieval

    readonly attribute unsigned long    length;
    DOMString      substringData(in unsigned long offset,
                                in unsigned long count)
                  raises(DOMException);
    void          appendData(in DOMString arg)
                  raises(DOMException);
    void          insertData(in unsigned long offset,
                            in DOMString arg)
```

dom.idl:

```

        raises(DOMException);
void      deleteData(in unsigned long offset,
                    in unsigned long count)
        raises(DOMException);
void      replaceData(in unsigned long offset,
                    in unsigned long count,
                    in DOMString arg)
        raises(DOMException);
};

interface Attr : Node {
    readonly attribute DOMString      name;
    readonly attribute boolean        specified;
    attribute DOMString               value;
    // raises(DOMException) on setting

    // Introduced in DOM Level 2:
    readonly attribute Element        ownerElement;
};

interface Element : Node {
    readonly attribute DOMString      tagName;
    DOMString      getAttribute(in DOMString name);
    void           setAttribute(in DOMString name,
                              in DOMString value)
        raises(DOMException);
    void           removeAttribute(in DOMString name)
        raises(DOMException);
    Attr           getAttributeNode(in DOMString name);
    Attr           setAttributeNode(in Attr newAttr)
        raises(DOMException);
    Attr           removeAttributeNode(in Attr oldAttr)
        raises(DOMException);
    NodeList       getElementsByTagName(in DOMString name);
    // Introduced in DOM Level 2:
    DOMString      getAttributeNS(in DOMString namespaceURI,
                              in DOMString localName);
    // Introduced in DOM Level 2:
    void           setAttributeNS(in DOMString namespaceURI,
                              in DOMString qualifiedName,
                              in DOMString value)
        raises(DOMException);
    // Introduced in DOM Level 2:
    void           removeAttributeNS(in DOMString namespaceURI,
                              in DOMString localName)
        raises(DOMException);
    // Introduced in DOM Level 2:
    Attr           getAttributeNodeNS(in DOMString namespaceURI,
                              in DOMString localName);
    // Introduced in DOM Level 2:
    Attr           setAttributeNodeNS(in Attr newAttr)
        raises(DOMException);
    // Introduced in DOM Level 2:
    NodeList       getElementsByTagNameNS(in DOMString namespaceURI,
                              in DOMString localName);
    // Introduced in DOM Level 2:
    boolean        hasAttribute(in DOMString name);
};
```

dom.idl:

```
// Introduced in DOM Level 2:
boolean          hasAttributeNS(in DOMString namespaceURI,
                                in DOMString localName);
};

interface Text : CharacterData {
    Text          splitText(in unsigned long offset)
                                raises(DOMException);
};

interface Comment : CharacterData {
};

interface CDATASection : Text {
};

interface DocumentType : Node {
    readonly attribute DOMString      name;
    readonly attribute NamedNodeMap   entities;
    readonly attribute NamedNodeMap   notations;
    // Introduced in DOM Level 2:
    readonly attribute DOMString      publicId;
    // Introduced in DOM Level 2:
    readonly attribute DOMString      systemId;
    // Introduced in DOM Level 2:
    readonly attribute DOMString      internalSubset;
};

interface Notation : Node {
    readonly attribute DOMString      publicId;
    readonly attribute DOMString      systemId;
};

interface Entity : Node {
    readonly attribute DOMString      publicId;
    readonly attribute DOMString      systemId;
    readonly attribute DOMString      notationName;
};

interface EntityReference : Node {
};

interface ProcessingInstruction : Node {
    readonly attribute DOMString      target;
    attribute DOMString              data;
    // raises(DOMException) on setting
};

interface DocumentFragment : Node {
};

interface Document : Node {
    readonly attribute DocumentType   doctype;
    readonly attribute DOMImplementation implementation;
    readonly attribute Element        documentElement;
    Element                          createElement(in DOMString tagName)
```

dom.idl:

```

                                raises(DOMException);
DocumentFragment  createDocumentFragment();
Text              createTextNode(in DOMString data);
Comment           createComment(in DOMString data);
CDATASection      createCDATASection(in DOMString data)
                                raises(DOMException);
ProcessingInstruction  createProcessingInstruction(in DOMString target,
                                                    in DOMString data)
                                raises(DOMException);
Attr              createAttribute(in DOMString name)
                                raises(DOMException);
EntityReference   createEntityReference(in DOMString name)
                                raises(DOMException);
NodeList          getElementsByTagName(in DOMString tagname);
// Introduced in DOM Level 2:
Node              importNode(in Node importedNode,
                              in boolean deep)
                                raises(DOMException);
// Introduced in DOM Level 2:
Element           createElementNS(in DOMString namespaceURI,
                                  in DOMString qualifiedName)
                                raises(DOMException);
// Introduced in DOM Level 2:
Attr              createAttributeNS(in DOMString namespaceURI,
                                    in DOMString qualifiedName)
                                raises(DOMException);
// Introduced in DOM Level 2:
NodeList          getElementsByTagNameNS(in DOMString namespaceURI,
                                          in DOMString localName);
// Introduced in DOM Level 2:
Element           getElementById(in DOMString elementId);
};
};
#endif // _DOM_IDL_
```

Appendix D: Java Language Binding

This appendix contains the complete Java Language [Java] binding for the Level 2 Document Object Model Core.

The Java files are also available as

<http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/java-binding.zip>

org/w3c/dom/DOMException.java:

```
package org.w3c.dom;

public class DOMException extends RuntimeException {
    public DOMException(short code, String message) {
        super(message);
        this.code = code;
    }
    public short    code;
    // ExceptionCode
    public static final short INDEX_SIZE_ERR           = 1;
    public static final short DOMSTRING_SIZE_ERR       = 2;
    public static final short HIERARCHY_REQUEST_ERR    = 3;
    public static final short WRONG_DOCUMENT_ERR       = 4;
    public static final short INVALID_CHARACTER_ERR    = 5;
    public static final short NO_DATA_ALLOWED_ERR      = 6;
    public static final short NO_MODIFICATION_ALLOWED_ERR = 7;
    public static final short NOT_FOUND_ERR            = 8;
    public static final short NOT_SUPPORTED_ERR        = 9;
    public static final short INUSE_ATTRIBUTE_ERR      = 10;
    public static final short INVALID_STATE_ERR        = 11;
    public static final short SYNTAX_ERR               = 12;
    public static final short INVALID_MODIFICATION_ERR = 13;
    public static final short NAMESPACE_ERR           = 14;
    public static final short INVALID_ACCESS_ERR       = 15;
}
```

org/w3c/dom/DOMImplementation.java:

```
package org.w3c.dom;

public interface DOMImplementation {
    public boolean hasFeature(String feature,
                               String version);

    public DocumentType createDocumentType(String qualifiedName,
                                           String publicId,
                                           String systemId)
        throws DOMException;

    public Document createDocument(String namespaceURI,
                                   String qualifiedName,
```

org/w3c/dom/DocumentFragment.java:

```
        DocumentType doctype)
        throws DOMException;

}
```

org/w3c/dom/DocumentFragment.java:

```
package org.w3c.dom;

public interface DocumentFragment extends Node {
}
```

org/w3c/dom/Document.java:

```
package org.w3c.dom;

public interface Document extends Node {
    public DocumentType getDoctype();

    public DOMImplementation getImplementation();

    public Element getDocumentElement();

    public Element createElement(String tagName)
        throws DOMException;

    public DocumentFragment createDocumentFragment();

    public Text createTextNode(String data);

    public Comment createComment(String data);

    public CDATASection createCDATASection(String data)
        throws DOMException;

    public ProcessingInstruction createProcessingInstruction(String target,
        String data)
        throws DOMException;

    public Attr createAttribute(String name)
        throws DOMException;

    public EntityReference createEntityReference(String name)
        throws DOMException;

    public NodeList getElementsByTagName(String tagname);

    public Node importNode(Node importedNode,
        boolean deep)
        throws DOMException;

    public Element createElementNS(String namespaceURI,
        String qualifiedName)
        throws DOMException;

    public Attr createAttributeNS(String namespaceURI,
```

org/w3c/dom/Node.java:

```
String qualifiedName)
throws DOMException;

public NodeList getElementsByTagNameNS(String namespaceURI,
String localName);

public Element getElementById(String elementId);
}
```

org/w3c/dom/Node.java:

```
package org.w3c.dom;

public interface Node {
    // NodeType
    public static final short ELEMENT_NODE           = 1;
    public static final short ATTRIBUTE_NODE         = 2;
    public static final short TEXT_NODE              = 3;
    public static final short CDATA_SECTION_NODE     = 4;
    public static final short ENTITY_REFERENCE_NODE  = 5;
    public static final short ENTITY_NODE            = 6;
    public static final short PROCESSING_INSTRUCTION_NODE = 7;
    public static final short COMMENT_NODE           = 8;
    public static final short DOCUMENT_NODE          = 9;
    public static final short DOCUMENT_TYPE_NODE     = 10;
    public static final short DOCUMENT_FRAGMENT_NODE = 11;
    public static final short NOTATION_NODE          = 12;

    public String getNodeName();

    public String getNodeValue()
        throws DOMException;
    public void setNodeValue(String nodeValue)
        throws DOMException;

    public short getNodeType();

    public Node getParentNode();

    public NodeList getChildNodes();

    public Node getFirstChild();

    public Node getLastChild();

    public Node getPreviousSibling();

    public Node getNextSibling();

    public NamedNodeMap getAttributes();

    public Document getOwnerDocument();

    public Node insertBefore(Node newChild,
        Node refChild)
```



```
        throws DOMException;

    public Node replaceChild(Node newChild,
                             Node oldChild)
        throws DOMException;

    public Node removeChild(Node oldChild)
        throws DOMException;

    public Node appendChild(Node newChild)
        throws DOMException;

    public boolean hasChildNodes();

    public Node cloneNode(boolean deep);

    public void normalize();

    public boolean isSupported(String feature,
                               String version);

    public String getNamespaceURI();

    public String getPrefix();
    public void setPrefix(String prefix)
        throws DOMException;

    public String getLocalName();

    public boolean hasAttributes();
}
```

org/w3c/dom/NodeList.java:

```
package org.w3c.dom;

public interface NodeList {
    public Node item(int index);

    public int getLength();
}
```

org/w3c/dom/NamedNodeMap.java:

```
package org.w3c.dom;

public interface NamedNodeMap {
    public Node getNamedItem(String name);

    public Node setNamedItem(Node arg)
        throws DOMException;

    public Node removeNamedItem(String name)
        throws DOMException;
}
```

```
public Node item(int index);

public int getLength();

public Node getNamedItemNS(String namespaceURI,
                           String localName);

public Node setNamedItemNS(Node arg)
                           throws DOMException;

public Node removeNamedItemNS(String namespaceURI,
                              String localName)
                              throws DOMException;

}
```

org/w3c/dom/CharacterData.java:

```
package org.w3c.dom;

public interface CharacterData extends Node {
    public String getData()
                           throws DOMException;
    public void setData(String data)
                       throws DOMException;

    public int getLength();

    public String substringData(int offset,
                               int count)
                               throws DOMException;

    public void appendData(String arg)
                        throws DOMException;

    public void insertData(int offset,
                          String arg)
                          throws DOMException;

    public void deleteData(int offset,
                          int count)
                          throws DOMException;

    public void replaceData(int offset,
                          int count,
                          String arg)
                          throws DOMException;
}
```

org/w3c/dom/Attr.java:

```
package org.w3c.dom;

public interface Attr extends Node {
    public String getName();

    public boolean getSpecified();

    public String getValue();
    public void setValue(String value)
        throws DOMException;

    public Element getOwnerElement();
}
```

org/w3c/dom/Element.java:

```
package org.w3c.dom;

public interface Element extends Node {
    public String getTagName();

    public String getAttribute(String name);

    public void setAttribute(String name,
        String value)
        throws DOMException;

    public void removeAttribute(String name)
        throws DOMException;

    public Attr getAttributeNode(String name);

    public Attr setAttributeNode(Attr newAttr)
        throws DOMException;

    public Attr removeAttributeNode(Attr oldAttr)
        throws DOMException;

    public NodeList getElementsByTagName(String name);

    public String getAttributeNS(String namespaceURI,
        String localName);

    public void setAttributeNS(String namespaceURI,
        String qualifiedName,
        String value)
        throws DOMException;

    public void removeAttributeNS(String namespaceURI,
        String localName)
        throws DOMException;

    public Attr getAttributeNodeNS(String namespaceURI,
```

org/w3c/dom/Text.java:

```
        String localName);

    public Attr setAttributeNodeNS(Attr newAttr)
        throws DOMException;

    public NodeList getElementsByTagNameNS(String namespaceURI,
        String localName);

    public boolean hasAttribute(String name);

    public boolean hasAttributeNS(String namespaceURI,
        String localName);
}
```

org/w3c/dom/Text.java:

```
package org.w3c.dom;

public interface Text extends CharacterData {
    public Text splitText(int offset)
        throws DOMException;
}
```

org/w3c/dom/Comment.java:

```
package org.w3c.dom;

public interface Comment extends CharacterData {
}
```

org/w3c/dom/CDATASection.java:

```
package org.w3c.dom;

public interface CDATASection extends Text {
}
```

org/w3c/dom/DocumentType.java:

```
package org.w3c.dom;

public interface DocumentType extends Node {
    public String getName();

    public NamedNodeMap getEntities();

    public NamedNodeMap getNotations();

    public String getPublicId();

    public String getSystemId();
}
```

```
    public String getInternalSubset();  
}
```

org/w3c/dom/Notation.java:

```
package org.w3c.dom;  
  
public interface Notation extends Node {  
    public String getPublicId();  
  
    public String getSystemId();  
}
```

org/w3c/dom/Entity.java:

```
package org.w3c.dom;  
  
public interface Entity extends Node {  
    public String getPublicId();  
  
    public String getSystemId();  
  
    public String getNotationName();  
}
```

org/w3c/dom/EntityReference.java:

```
package org.w3c.dom;  
  
public interface EntityReference extends Node {  
}
```

org/w3c/dom/ProcessingInstruction.java:

```
package org.w3c.dom;  
  
public interface ProcessingInstruction extends Node {  
    public String getTarget();  
  
    public String getData();  
    public void setData(String data)  
        throws DOMException;  
}
```

Appendix E: ECMAScript Language Binding

This appendix contains the complete ECMAScript [ECMAScript] binding for the Level 2 Document Object Model Core definitions.

Note: Exceptions handling is only supported by ECMAScript implementation conformant with the Standard ECMA-262 3rd. Edition ([ECMAScript]).

Prototype Object **DOMException**

The **DOMException** class has the following constants:

DOMException.INDEX_SIZE_ERR

This constant is of type **Number** and its value is **1**.

DOMException.DOMSTRING_SIZE_ERR

This constant is of type **Number** and its value is **2**.

DOMException.HIERARCHY_REQUEST_ERR

This constant is of type **Number** and its value is **3**.

DOMException.WRONG_DOCUMENT_ERR

This constant is of type **Number** and its value is **4**.

DOMException.INVALID_CHARACTER_ERR

This constant is of type **Number** and its value is **5**.

DOMException.NO_DATA_ALLOWED_ERR

This constant is of type **Number** and its value is **6**.

DOMException.NO_MODIFICATION_ALLOWED_ERR

This constant is of type **Number** and its value is **7**.

DOMException.NOT_FOUND_ERR

This constant is of type **Number** and its value is **8**.

DOMException.NOT_SUPPORTED_ERR

This constant is of type **Number** and its value is **9**.

DOMException.INUSE_ATTRIBUTE_ERR

This constant is of type **Number** and its value is **10**.

DOMException.INVALID_STATE_ERR

This constant is of type **Number** and its value is **11**.

DOMException.SYNTAX_ERR

This constant is of type **Number** and its value is **12**.

DOMException.INVALID_MODIFICATION_ERR

This constant is of type **Number** and its value is **13**.

DOMException.NAMESPACE_ERR

This constant is of type **Number** and its value is **14**.

DOMException.INVALID_ACCESS_ERR

This constant is of type **Number** and its value is **15**.

Object **DOMException**

The **DOMException** object has the following properties:

code

This property is of type **Number**.

Object DOMImplementation

The **DOMImplementation** object has the following methods:

hasFeature(feature, version)

This method returns a **Boolean**.

The **feature** parameter is of type **String**.

The **version** parameter is of type **String**.

createDocumentType(qualifiedName, publicId, systemId)

This method returns a **DocumentType** object.

The **qualifiedName** parameter is of type **String**.

The **publicId** parameter is of type **String**.

The **systemId** parameter is of type **String**.

This method can raise a **DOMException** object.

createDocument(namespaceURI, qualifiedName, doctype)

This method returns a **Document** object.

The **namespaceURI** parameter is of type **String**.

The **qualifiedName** parameter is of type **String**.

The **doctype** parameter is a **DocumentType** object.

This method can raise a **DOMException** object.

Object DocumentFragment

DocumentFragment has the all the properties and methods of the **Node** object as well as the properties and methods defined below.

Object Document

Document has the all the properties and methods of the **Node** object as well as the properties and methods defined below.

The **Document** object has the following properties:

doctype

This read-only property is a **DocumentType** object.

implementation

This read-only property is a **DOMImplementation** object.

documentElement

This read-only property is a **Element** object.

The **Document** object has the following methods:

createElement(tagName)

This method returns a **Element** object.

The **tagName** parameter is of type **String**.

This method can raise a **DOMException** object.

createDocumentFragment()

This method returns a **DocumentFragment** object.

createTextNode(data)

This method returns a **Text** object.

The **data** parameter is of type **String**.

createComment(data)

This method returns a **Comment** object.

The **data** parameter is of type **String**.

createCDATASection(data)

This method returns a **CDATASection** object.

The **data** parameter is of type **String**.

This method can raise a **DOMException** object.

createProcessingInstruction(target, data)

This method returns a **ProcessingInstruction** object.

The **target** parameter is of type **String**.

The **data** parameter is of type **String**.

This method can raise a **DOMException** object.

createAttribute(name)

This method returns a **Attr** object.

The **name** parameter is of type **String**.

This method can raise a **DOMException** object.

createEntityReference(name)

This method returns a **EntityReference** object.

The **name** parameter is of type **String**.

This method can raise a **DOMException** object.

getElementsByTagName(tagname)

This method returns a **NodeList** object.

The **tagname** parameter is of type **String**.

importNode(importedNode, deep)

This method returns a **Node** object.

The **importedNode** parameter is a **Node** object.

The **deep** parameter is of type **Boolean**.

This method can raise a **DOMException** object.

createElementNS(namespaceURI, qualifiedName)

This method returns a **Element** object.

The **namespaceURI** parameter is of type **String**.

The **qualifiedName** parameter is of type **String**.

This method can raise a **DOMException** object.

createAttributeNS(namespaceURI, qualifiedName)

This method returns a **Attr** object.

The **namespaceURI** parameter is of type **String**.

The **qualifiedName** parameter is of type **String**.

This method can raise a **DOMException** object.

getElementsByTagNameNS(namespaceURI, localName)

This method returns a **NodeList** object.

The **namespaceURI** parameter is of type **String**.

The **localName** parameter is of type **String**.

getElementById(elementId)

This method returns a **Element** object.

The **elementId** parameter is of type **String**.

Prototype Object **Node**

The **Node** class has the following constants:

Node.ELEMENT_NODE

This constant is of type **Number** and its value is **1**.

Node.ATTRIBUTE_NODE

This constant is of type **Number** and its value is **2**.

Node.TEXT_NODE

This constant is of type **Number** and its value is **3**.

Node.CDATA_SECTION_NODE

This constant is of type **Number** and its value is **4**.

Node.ENTITY_REFERENCE_NODE

This constant is of type **Number** and its value is **5**.

Node.ENTITY_NODE

This constant is of type **Number** and its value is **6**.

Node.PROCESSING_INSTRUCTION_NODE

This constant is of type **Number** and its value is **7**.

Node.COMMENT_NODE

This constant is of type **Number** and its value is **8**.

Node.DOCUMENT_NODE

This constant is of type **Number** and its value is **9**.

Node.DOCUMENT_TYPE_NODE

This constant is of type **Number** and its value is **10**.

Node.DOCUMENT_FRAGMENT_NODE

This constant is of type **Number** and its value is **11**.

Node.NOTATION_NODE

This constant is of type **Number** and its value is **12**.

Object Node

The **Node** object has the following properties:

nodeName

This read-only property is of type **String**.

nodeValue

This property is of type **String**, can raise a **DOMException** object on setting and can raise a **DOMException** object on retrieval.

nodeType

This read-only property is of type **Number**.

parentNode

This read-only property is a **Node** object.

childNodes

This read-only property is a **NodeList** object.

firstChild

This read-only property is a **Node** object.

lastChild

This read-only property is a **Node** object.

previousSibling

This read-only property is a **Node** object.

nextSibling

This read-only property is a **Node** object.

attributes

This read-only property is a **NamedNodeMap** object.

ownerDocument

This read-only property is a **Document** object.

namespaceURI

This read-only property is of type **String**.

prefix

This property is of type **String** and can raise a **DOMException** object on setting.

localName

This read-only property is of type **String**.

The **Node** object has the following methods:

insertBefore(newChild, refChild)

This method returns a **Node** object.

The **newChild** parameter is a **Node** object.

The **refChild** parameter is a **Node** object.

This method can raise a **DOMException** object.

replaceChild(newChild, oldChild)

This method returns a **Node** object.

The **newChild** parameter is a **Node** object.

The **oldChild** parameter is a **Node** object.

This method can raise a **DOMException** object.

removeChild(oldChild)

This method returns a **Node** object.

The **oldChild** parameter is a **Node** object.

This method can raise a **DOMException** object.

appendChild(newChild)

This method returns a **Node** object.

The **newChild** parameter is a **Node** object.

This method can raise a **DOMException** object.

hasChildNodes()

This method returns a **Boolean**.

cloneNode(deep)

This method returns a **Node** object.

The **deep** parameter is of type **Boolean**.

normalize()

This method has no return value.

isSupported(feature, version)

This method returns a **Boolean**.

The **feature** parameter is of type **String**.

The **version** parameter is of type **String**.

hasAttributes()

This method returns a **Boolean**.

Object **NodeList**

The **NodeList** object has the following properties:

length

This read-only property is of type **Number**.

The **NodeList** object has the following methods:

item(index)

This method returns a **Node** object.

The **index** parameter is of type **Number**.

Note: This object can also be dereferenced using square bracket notation (e.g. `obj[1]`). Dereferencing with an integer **index** is equivalent to invoking the **item** method with that index.

Object **NamedNodeMap**

The **NamedNodeMap** object has the following properties:

length

This read-only property is of type **Number**.

The **NamedNodeMap** object has the following methods:

getNamedItem(name)

This method returns a **Node** object.

The **name** parameter is of type **String**.

setNamedItem(arg)

This method returns a **Node** object.

The **arg** parameter is a **Node** object.

This method can raise a **DOMException** object.

removeNamedItem(name)

This method returns a **Node** object.

The **name** parameter is of type **String**.

This method can raise a **DOMException** object.

item(index)

This method returns a **Node** object.

The **index** parameter is of type **Number**.

Note: This object can also be dereferenced using square bracket notation (e.g. `obj[1]`).

Dereferencing with an integer **index** is equivalent to invoking the **item** method with that index.

getNamedItemNS(namespaceURI, localName)

This method returns a **Node** object.

The **namespaceURI** parameter is of type **String**.

The **localName** parameter is of type **String**.

setNamedItemNS(arg)

This method returns a **Node** object.

The **arg** parameter is a **Node** object.

This method can raise a **DOMException** object.

removeNamedItemNS(namespaceURI, localName)

This method returns a **Node** object.

The **namespaceURI** parameter is of type **String**.

The **localName** parameter is of type **String**.

This method can raise a **DOMException** object.

Object **CharacterData**

CharacterData has all the properties and methods of the **Node** object as well as the properties and methods defined below.

The **CharacterData** object has the following properties:

data

This property is of type **String**, can raise a **DOMException** object on setting and can raise a **DOMException** object on retrieval.

length

This read-only property is of type **Number**.

The **CharacterData** object has the following methods:

substringData(offset, count)

This method returns a **String**.

The **offset** parameter is of type **Number**.

The **count** parameter is of type **Number**.

This method can raise a **DOMException** object.

appendData(arg)

This method has no return value.

The **arg** parameter is of type **String**.

This method can raise a **DOMException** object.

insertData(offset, arg)

This method has no return value.

The **offset** parameter is of type **Number**.

The **arg** parameter is of type **String**.

This method can raise a **DOMException** object.

deleteData(offset, count)

This method has no return value.

The **offset** parameter is of type **Number**.

The **count** parameter is of type **Number**.

This method can raise a **DOMException** object.

replaceData(offset, count, arg)

This method has no return value.

The **offset** parameter is of type **Number**.

The **count** parameter is of type **Number**.

The **arg** parameter is of type **String**.

This method can raise a **DOMException** object.

Object **Attr**

Attr has the all the properties and methods of the **Node** object as well as the properties and methods defined below.

The **Attr** object has the following properties:

name

This read-only property is of type **String**.

specified

This read-only property is of type **Boolean**.

value

This property is of type **String** and can raise a **DOMException** object on setting.

ownerElement

This read-only property is a **Element** object.

Object **Element**

Element has the all the properties and methods of the **Node** object as well as the properties and methods defined below.

The **Element** object has the following properties:

tagName

This read-only property is of type **String**.

The **Element** object has the following methods:

getAttribute(name)

This method returns a **String**.

The **name** parameter is of type **String**.

setAttribute(name, value)

This method has no return value.

The **name** parameter is of type **String**.

The **value** parameter is of type **String**.

This method can raise a **DOMException** object.

removeAttribute(name)

This method has no return value.

The **name** parameter is of type **String**.

This method can raise a **DOMException** object.

getAttributeNode(name)

This method returns a **Attr** object.

The **name** parameter is of type **String**.

setAttributeNode(newAttr)

This method returns a **Attr** object.

The **newAttr** parameter is a **Attr** object.

This method can raise a **DOMException** object.

removeAttributeNode(oldAttr)

This method returns a **Attr** object.

The **oldAttr** parameter is a **Attr** object.

This method can raise a **DOMException** object.

getElementsByTagName(name)

This method returns a **NodeList** object.

The **name** parameter is of type **String**.

getAttributeNS(namespaceURI, localName)

This method returns a **String**.

The **namespaceURI** parameter is of type **String**.

The **localName** parameter is of type **String**.

setAttributeNS(namespaceURI, qualifiedName, value)

This method has no return value.

The **namespaceURI** parameter is of type **String**.

The **qualifiedName** parameter is of type **String**.

The **value** parameter is of type **String**.

This method can raise a **DOMException** object.

removeAttributeNS(namespaceURI, localName)

This method has no return value.

The **namespaceURI** parameter is of type **String**.

The **localName** parameter is of type **String**.

This method can raise a **DOMException** object.

getAttributeNodeNS(namespaceURI, localName)

This method returns a **Attr** object.

The **namespaceURI** parameter is of type **String**.

The **localName** parameter is of type **String**.

setAttributeNodeNS(newAttr)

This method returns a **Attr** object.

The **newAttr** parameter is a **Attr** object.

This method can raise a **DOMException** object.

getElementsByTagNameNS(namespaceURI, localName)

This method returns a **NodeList** object.

The **namespaceURI** parameter is of type **String**.

The **localName** parameter is of type **String**.

hasAttribute(name)

This method returns a **Boolean**.

The **name** parameter is of type **String**.

hasAttributeNS(namespaceURI, localName)

This method returns a **Boolean**.

The **namespaceURI** parameter is of type **String**.

The **localName** parameter is of type **String**.

Object Text

Text has the all the properties and methods of the **CharacterData** object as well as the properties and methods defined below.

The **Text** object has the following methods:

splitText(offset)

This method returns a **Text** object.

The **offset** parameter is of type **Number**.

This method can raise a **DOMException** object.

Object Comment

Comment has the all the properties and methods of the **CharacterData** object as well as the properties and methods defined below.

Object CDATASection

CDATASection has the all the properties and methods of the **Text** object as well as the properties and methods defined below.

Object DocumentType

DocumentType has the all the properties and methods of the **Node** object as well as the properties and methods defined below.

The **DocumentType** object has the following properties:

name

This read-only property is of type **String**.

entities

This read-only property is a **NamedNodeMap** object.

notations

This read-only property is a **NamedNodeMap** object.

publicId

This read-only property is of type **String**.

systemId

This read-only property is of type **String**.

internalSubset

This read-only property is of type **String**.

Object Notation

Notation has the all the properties and methods of the **Node** object as well as the properties and methods defined below.

The **Notation** object has the following properties:

publicId

This read-only property is of type **String**.

systemId

This read-only property is of type **String**.

Object Entity

Entity has the all the properties and methods of the **Node** object as well as the properties and methods defined below.

The **Entity** object has the following properties:

publicId

This read-only property is of type **String**.

systemId

This read-only property is of type **String**.

notationName

This read-only property is of type **String**.

Object EntityReference

EntityReference has the all the properties and methods of the **Node** object as well as the properties and methods defined below.

Object ProcessingInstruction

ProcessingInstruction has the all the properties and methods of the **Node** object as well as the properties and methods defined below.

The **ProcessingInstruction** object has the following properties:

target

This read-only property is of type **String**.

data

This property is of type **String** and can raise a **DOMException** object on setting.

Appendix F: Acknowledgements

Many people contributed to this specification, including members of the DOM Working Group and the DOM Interest Group. We especially thank the following:

Lauren Wood (SoftQuad Software Inc., *chair*), Andrew Watson (Object Management Group), Andy Heninger (IBM), Arnaud Le Hors (W3C and IBM), Ben Chang (Oracle), Bill Smith (Sun), Bill Shea (Merrill Lynch), Bob Sutor (IBM), Chris Lovett (Microsoft), Chris Wilson (Microsoft), David Brownell (Sun), David Singer (IBM), Don Park (invited), Eric Vasilik (Microsoft), Gavin Nicol (INSO), Ian Jacobs (W3C), James Clark (invited), James Davidson (Sun), Jared Sorensen (Novell), Joe Kesselman (IBM), Joe Lapp (webMethods), Joe Marini (Macromedia), Johnny Stenback (Netscape), Jonathan Marsh (Microsoft), Jonathan Robie (Texcel Research and Software AG), Kim Adamson-Sharpe (SoftQuad Software Inc.), Laurence Cable (Sun), Mark Davis (IBM), Mark Scardina (Oracle), Martin Dürst (W3C), Mick Goulish (Software AG), Mike Champion (Arbortext and Software AG), Miles Sabin (Cromwell Media), Patti Lutsky (Arbortext), Paul Grosso (Arbortext), Peter Sharpe (SoftQuad Software Inc.), Phil Karlton (Netscape), Philippe Le Hégarret (W3C, *W3C team contact*), Ramesh Lekshmyanarayanan (Merrill Lynch), Ray Whitmer (iMall, Excite@Home and Netscape), Rich Rollman (Microsoft), Rick Gessner (Netscape), Scott Isaacs (Microsoft), Sharon Adler (INSO), Steve Byrne (JavaSoft), Tim Bray (invited), Tom Pixley (Netscape), Vidur Apparao (Netscape), Vinod Anupam (Lucent).

Thanks to all those who have helped to improve this specification by sending suggestions and corrections.

F.1: Production Systems

This specification was written in XML. The HTML, OMG IDL, Java and ECMA Script bindings were all produced automatically.

Thanks to Joe English, author of cost, which was used as the basis for producing DOM Level 1. Thanks also to Gavin Nicol, who wrote the scripts which run on top of cost. Arnaud Le Hors and Philippe Le Hégarret maintained the scripts.

For DOM Level 2, we used Xerces as the basis DOM implementation and wish to thank the authors. Philippe Le Hégarret and Arnaud Le Hors wrote the Java programs which are the DOM application.

Thanks also to Jan Kärman, author of html2ps, which we use in creating the PostScript version of the specification.

Glossary

Editors

Arnaud Le Hors, IBM
 Lauren Wood, SoftQuad Software Inc.
 Robert S. Sutor, IBM (for DOM Level 1)

Several of the following term definitions have been borrowed or modified from similar definitions in other W3C or standards documents. See the links within the definitions for more information.

16-bit unit

The base unit of a `DOMString` [p.17] . This indicates that indexing on a `DOMString` occurs in units of 16 bits. This must not be misunderstood to mean that a `DOMString` can store arbitrary 16-bit units. A `DOMString` is a character string encoded in UTF-16; this means that the restrictions of UTF-16 as well as the other relevant restrictions on character strings must be maintained. A single character, for example in the form of a numeric character reference, may correspond to one or two 16-bit units.

For more information, see [Unicode] and [ISO/IEC 10646].

ancestor

An *ancestor* node of any node A is any node above A in a tree model of a document, where "above" means "toward the root."

API

An *API* is an application programming interface, a set of functions or *methods* used to access some functionality.

child

A *child* is an immediate *descendant* node of a node.

client application

A [client] application is any software that uses the Document Object Model programming interfaces provided by the hosting implementation to accomplish useful work. Some examples of client applications are scripts within an HTML or XML document.

COM

COM is Microsoft's Component Object Model [COM], a technology for building applications from binary software components.

convenience

A *convenience method* is an operation on an object that could be accomplished by a program consisting of more basic operations on the object. Convenience *methods* are usually provided to make the API easier and simpler to use or to allow specific programs to create more optimized implementations for common operations. A similar definition holds for a *convenience property*.

data model

A *data model* is a collection of descriptions of data structures and their contained fields, together with the operations or functions that manipulate them.

descendant

A *descendant* node of any node A is any node below A in a tree model of a document, where "above" means "toward the root."

ECMAScript

The programming language defined by the ECMA-262 standard [ECMAScript]. As stated in the standard, the originating technology for ECMAScript was JavaScript [JavaScript]. Note that in the ECMAScript Language binding, the word "property" is used in the same sense as the IDL term "attribute."

element

Each document contains one or more elements, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements by an empty-element tag. Each element has a type, identified by name, and may have a set of attributes. Each attribute has a name and a value. See *Logical Structures* in XML [XML].

information item

An information item is an abstract representation of some component of an XML document. See the [Infoset] for details.

hosting implementation

A [hosting] implementation is a software module that provides an implementation of the DOM interfaces so that a client application can use them. Some examples of hosting implementations are browsers, editors and document repositories.

HTML

The HyperText Markup Language (*HTML*) is a simple markup language used to create hypertext documents that are portable from one platform to another. HTML documents are SGML documents with generic semantics that are appropriate for representing information from a wide range of applications. [HTML4.0]

inheritance

In object-oriented programming, the ability to create new classes (or interfaces) that contain all the methods and properties of another class (or interface), plus additional methods and properties. If class (or interface) D inherits from class (or interface) B, then D is said to be *derived* from B. B is said to be a *base* class (or interface) for D. Some programming languages allow for multiple inheritance, that is, inheritance from more than one class or interface.

interface

An *interface* is a declaration of a set of *methods* with no information given about their implementation. In object systems that support interfaces and inheritance, interfaces can usually inherit from one another.

language binding

A programming *language binding* for an IDL specification is an implementation of the interfaces in the specification for the given language. For example, a Java language binding for the Document Object Model IDL specification would implement the concrete Java classes that provide the functionality exposed by the interfaces.

local name

A *local name* is the local part of a *qualified name*. This is called the local part in Namespaces in XML [Namespaces].

method

A *method* is an operation or function that is associated with an object and is allowed to manipulate the object's data.

model

A *model* is the actual data representation for the information at hand. Examples are the structural model and the style model representing the parse structure and the style information associated with a

document. The model might be a tree, or a directed graph, or something else.

namespace prefix

A *namespace prefix* is a string that associates an element or attribute name with a *namespace URI* in XML. See namespace prefix in Namespaces in XML [Namespaces].

namespace URI

A *namespace URI* is a URI that identifies an *XML namespace*. Strictly speaking, this actually is a *namespace URI reference*. This is called the namespace name in Namespaces in XML [Namespaces].

object model

An *object model* is a collection of descriptions of classes or interfaces, together with their member data, member functions, and class-static operations.

parent

A *parent* is an immediate *ancestor* node of a node.

qualified name

A *qualified name* is the name of an element or attribute defined as the concatenation of a *local name* (as defined in this specification), optionally preceded by a *namespace prefix* and colon character. See *Qualified Names* in Namespaces in XML [Namespaces].

readonly node

A *readonly node* is a node that is immutable. This means its list of children, its content, and its attributes, when it is an element, cannot be changed in any way. However, a readonly node can possibly be moved, when it is not itself contained in a readonly node.

root node

The *root node* is the unique node that is not a *child* of any other node. All other nodes are children or other descendants of the root node.

sibling

Two nodes are *siblings* if and only if they have the same *parent* node.

string comparison

When string matching is required, it is to occur as though the comparison was between 2 sequences of code points from the Unicode 3.0 standard [Unicode].

token

An information item such as an XML Name which has been *tokenized* [p.99] .

tokenized

The description given to various information items (for example, attribute values of various types, but not including the StringType CDATA) after having been processed by the XML processor. The process includes stripping leading and trailing white space, and replacing multiple space characters by one. See the definition of tokenized type.

well-formed document

A document is *well-formed* if it is tag valid and entities are limited to single elements (i.e., single sub-trees). See *Well-Formed XML Documents* in XML [XML].

XML

Extensible Markup Language (*XML*) is an extremely simple dialect of SGML. The goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML [XML] has been designed for ease of implementation and for interoperability with both SGML and HTML.

XML name

See *XML name* in the XML specification [XML].

XML namespace

An *XML namespace* is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute names. [Namespaces]

References

For the latest version of any W3C specification please consult the list of W3C Technical Reports available at <http://www.w3.org/TR>.

H.1: Normative references

Charmod

W3C (World Wide Web Consortium) Character Model for the World Wide Web, November 1999. Available at <http://www.w3.org/TR/1999/WD-charmod-19991129>

ECMAScript

ECMA (European Computer Manufacturers Association) ECMAScript Language Specification. Available at <http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM>

HTML4.0

W3C (World Wide Web Consortium) HTML 4.0 Specification, April 1998. Available at <http://www.w3.org/TR/1998/REC-html40-19980424>

ISO/IEC 10646

ISO (International Organization for Standardization). ISO/IEC 10646-1:2000 (E). Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane. [Geneva]: International Organization for Standardization.

Java

Sun Microsystems Inc. The Java Language Specification, James Gosling, Bill Joy, and Guy Steele, September 1996. Available at <http://java.sun.com/docs/books/jls>

Namespaces

W3C (World Wide Web Consortium) Namespaces in XML, January 1999. Available at <http://www.w3.org/TR/1999/REC-xml-names-19990114>

OMGIDL

OMG (Object Management Group) IDL (Interface Definition Language) defined in The Common Object Request Broker: Architecture and Specification, version 2.3.1, October 1999. Available from <http://www.omg.org/>

RFC2396

IETF (Internet Engineering Task Force) RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, eds. T. Berners-Lee, R. Fielding, L. Masinter. August 1998. Available at <http://www.ietf.org/rfc/rfc2396.txt>

Unicode

The Unicode Consortium. The Unicode Standard, Version 3.0., February 2000. Available at <http://www.unicode.org/unicode/standard/versions/Unicode3.0.html>.

XML

W3C (World Wide Web Consortium) Extensible Markup Language (XML) 1.0, February 1998. Available at <http://www.w3.org/TR/1998/REC-xml-19980210>

H.2: Informative references

DOM Level 2 CSS

W3C (World Wide Web Consortium) Document Object Model Level 2 CSS. Available at <http://www.w3.org/TR/DOM-Level-2-Style/css>

COM

Microsoft Corp. The Component Object Model. Available at <http://www.microsoft.com/com>

CORBA

OMG (Object Management Group) The Common Object Request Broker: Architecture and Specification, version 2.3.1, October 1999. Available from <http://www.omg.org/>

DOM Level 1

W3C (World Wide Web Consortium) DOM Level 1 Specification, October 1998. Available at <http://www.w3.org/TR/REC-DOM-Level-1>

DOM Level 2 HTML

W3C (World Wide Web Consortium) Document Object Model Level 2 HTML Specification. Available at <http://www.w3.org/TR/DOM-Level-2-HTML>

DOM Level 2 Events

W3C (World Wide Web Consortium) Document Object Model Level 2 Events Specification. Available at <http://www.w3.org/TR/DOM-Level-2-Events>

InfoSet

W3C (World Wide Web Consortium) XML Information Set, December 1999. Available at <http://www.w3.org/TR/xml-infoSet>

JavaIDL

Sun Microsystems Inc. Java IDL. Available at <http://java.sun.com/products/jdk/1.2/docs/guide/idl>

JavaScript

Netscape Communications Corp. JavaScript Resources. Available at <http://developer.netscape.com/tech/javascript/resources.html>

JScript

Microsoft Corp. JScript Resources. Available at <http://msdn.microsoft.com/scripting/default.htm>

MIDL

Microsoft Corp. MIDL Language Reference. Available at http://msdn.microsoft.com/library/psdk/midl/mi-laref_1r1h.htm

DOM Level 2 Style Sheets

W3C (World Wide Web Consortium) Document Object Model Level 2 Style Sheets. Available at <http://www.w3.org/TR/DOM-Level-2-Style/stylesheets>

DOM Level 2 Traversal

W3C (World Wide Web Consortium) Document Object Model Level 2 Traversal. Available at <http://www.w3.org/TR/DOM-Level-2-Traversal-Range/traversal>

DOM Level 2 Range

W3C (World Wide Web Consortium) Document Object Model Level 2 Range. Available at <http://www.w3.org/TR/DOM-Level-2-Traversal-Range/ranges>

DOM Level 2 Views

W3C (World Wide Web Consortium) Document Object Model Level 2 Views Specification. Available at <http://www.w3.org/TR/DOM-Level-2-Views>

XPointer

W3C (World Wide Web Consortium) XML Pointer Language (XPointer), June 2000. Available at <http://www.w3.org/TR/xptr>

Index

16-bit unit 17, 18, 47, 49, 49, 49, 50, 60, 97

ancestor 40, 42, 38, 97

appendData

attributes

CDATA_SECTION_NODE

Charmod 18, 101

client application 9, 97

Comment

CORBA 9, 102

createCDATASection

createDocumentFragment

createElementNS

createTextNode

data 48, 66

descendant 19, 32, 55, 55, 64, 65, 97

DOCUMENT_FRAGMENT_NODE

documentElement

DOM Level 1 12, 102

DOM Level 2 HTML 12, 20, 61, 102

DOM Level 2 Traversal 12, 102

DOMImplementation

DOMTimeStamp

ECMAScript 9, 16, 98, 101

entities

ENTITY_REFERENCE_NODE

firstChild

API 9, 9, 11, 17, 17, 97

Attr

CDATASection

child 15, 19, 97

cloneNode

COMMENT_NODE

createAttribute

createComment

createDocumentType

createEntityReference

data model 9, 97

doctype

DOCUMENT_NODE

DocumentFragment

DOM Level 2 CSS 12, 102

DOM Level 2 Range 12, 102

DOM Level 2 Views 12, 102

DOMString

Element 52, 15, 16, 18, 19, 98

Entity

EntityReference

appendChild

ATTRIBUTE_NODE

CharacterData

childNodes

COM 9, 17, 97, 102

convenience 26, 52, 97

createAttributeNS

createDocument

createElement

createProcessingInstruction

deleteData

Document

DOCUMENT_TYPE_NODE

DocumentType

DOM Level 2 Events 12, 102

DOM Level 2 Style Sheets 12, 102

DOMException

DOMSTRING_SIZE_ERR

ELEMENT_NODE

ENTITY_NODE

Index

getAttribute	getAttributeNode	getAttributeNodeNS
getAttributeNS	getElementById	getElementsByTagName 31, 55
getElementsByTagNameNS 32, 55	getNamedItem	getNamedItemNS
hasAttribute	hasAttributeNS	hasAttributes
hasChildNodes	hasFeature	HIERARCHY_REQUEST_ERR
hosting implementation 12, 98	HTML 9, 98	HTML4.0 98, 101
implementation	importNode	INDEX_SIZE_ERR
information item 60, 98	Infoset 9, 11, 98, 102	inheritance 17, 98
insertBefore	insertData	interface 9, 98
internalSubset	INUSE_ATTRIBUTE_ERR	INVALID_ACCESS_ERR
INVALID_CHARACTER_ERR	INVALID_MODIFICATION_ERR	INVALID_STATE_ERR
ISO/IEC 10646 17, 97, 101	isSupported	item 43, 45
Java 9, 101	JavaIDL 9, 102	JavaScript 9, 98, 102
JScript 9, 102		
language binding 9, 98	lastChild	length 43, 44, 49
live 16, 43, 44	local name 29, 27, 32, 45, 46, 54, 57, 55, 59, 55, 56, 98	localName
method 12, 98	MIDL 9, 102	model 9, 98
name 52, 63	NamedNodeMap	namespace prefix 19, 30, 38, 64, 65, 99
namespace URI 19, 22, 29, 27, 32, 37, 45, 46, 54, 58, 57, 55, 59, 55, 56, 65, 99	NAMESPACE_ERR	Namespace 19, 22, 29, 37, 38, 98, 99, 99, 99, 100, 101
namespaceURI	nextSibling	NO_DATA_ALLOWED_ERR
NO_MODIFICATION_ALLOWED_ERR	Node	NodeList
nodeName	nodeType	nodeValue
normalize	NOT_FOUND_ERR	NOT_SUPPORTED_ERR
Notation	NOTATION_NODE	notationName
notations		

Index

object model 9, 11, 99	OMGIDL 9, 17, 101	ownerDocument
ownerElement		
parent 38, 99	parentNode	prefix
previousSibling	PROCESSING_INSTRUCTION_NODE	ProcessingInstruction
publicId 63, 64, 65		
qualified name 19, 23, 22, 29, 27, 38, 37, 58, 99		
readonly node 39, 64, 64, 65, 99	removeAttribute	removeAttributeNode
removeAttributeNS	removeChild	removeNamedItem
removeNamedItemNS	replaceChild	replaceData
RFC2396 100, 101	root node 25, 99	
setAttribute	setAttributeNode	setAttributeNodeNS
setAttributeNS	setNamedItem	setNamedItemNS
sibling 24, 60, 99	specified	splitText
string comparison 18, 19, 99	substringData	SYNTAX_ERR
systemId 63, 64, 65		
tagName	target	Text
TEXT_NODE	token 66, 99	tokenized 51, 99
Unicode 17, 97, 99, 101		
value		
well-formed document 24, 99	WRONG_DOCUMENT_ERR	
XML 9, 64, 99, 98, 99, 99, 101	XML name 24, 99	XML namespace 19, 100
XPointer 41, 103		



Document Object Model (DOM) Level 2 Core Specification

Version 1.0

W3C Proposed Recommendation 27 September, 2000

This version:

<http://www.w3.org/TR/2000/PR-DOM-Level-2-Core-20000927>

([PostScript file](#), [PDF file](#), [plain text](#), [ZIP file](#))

Latest version:

<http://www.w3.org/TR/DOM-Level-2-Core>

Previous version:

<http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510>

Editors:

Mark Davis, *IBM*

Arnaud Le Hors, *W3C team contact until October 1999, then IBM*

Philippe Le Hégaré, *W3C, team contact (from November 1999)*

Jonathan Robie, *Texcel Research and Software AG*

Lauren Wood, *SoftQuad Software Inc., chair*

[Copyright](#) © 2000 [W3C](#)® ([MIT](#), [INRIA](#), [Keio](#)), All Rights Reserved. W3C [liability](#), [trademark](#), [document use](#) and [software licensing](#) rules apply.

Abstract

This specification defines the Document Object Model Level 2 Core, a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content and structure of documents. The Document Object Model Level 2 Core builds on the Document Object Model Level 1

Core.

The DOM Level 2 Core is made of a set of core interfaces to create and manipulate the structure and contents of a document. The Core also contains specialized interfaces dedicated to XML.

Status of this document

This is a W3C [Proposed Recommendation](#) for review by W3C members and other interested parties. W3C Advisory Committee Members are invited to send formal comments, visible only to the W3C Team, to dom-review@w3.org until October 25, 2000.

Comments on this document are invited and are to be sent to the public mailing list www-dom@w3.org. An archive is available at <http://lists.w3.org/Archives/Public/www-dom/>.

Publication as a Proposed Recommendation does not imply endorsement by the W3C membership. This is still a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite W3C Proposed Recommendations as other than "work in progress."

This document has been produced as part of the [W3C DOM Activity](#). The authors of this document are the DOM WG members. Different modules of the Document Object Model have different editors.

A list of [current W3C Recommendations and other technical documents](#) can be found at <http://www.w3.org/TR>.

Table of contents

- [Expanded Table of Contents](#)
- [Copyright Notice](#)
- [What is the Document Object Model?](#)
- [1. Document Object Model Core](#)
- [Appendix A: Changes](#)
- [Appendix B: Accessing code point boundaries](#)
- [Appendix C: IDL Definitions](#)
- [Appendix D: Java Language Binding](#)
- [Appendix E: ECMA Script Language Binding](#)
- [Appendix F: Acknowledgements](#)
- [Glossary](#)
- [References](#)
- [Index](#)



Document Object Model (DOM) Level 2 Specification

Version 1.0

W3C Candidate Recommendation *10 May, 2000*

This version:

<http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510>
([PostScript file](#), [PDF file](#), [plain text](#), [ZIP file](#))

Latest version:

<http://www.w3.org/TR/DOM-Level-2>

Previous version:

<http://www.w3.org/TR/2000/CR-DOM-Level-2-20000307>

Editors:

Lauren Wood, *SoftQuad Software Inc., chair*
Arnaud Le Hors, *W3C staff contact until October 1999, then IBM*
Vidur Apparao, *Netscape Communications Corporation*
Laurence Cable, *Sun*
Mike Champion, *Arbortext and Software AG*
Mark Davis, *IBM*
Joe Kesselman, *IBM*
Philippe Le Hégarret, *W3C, staff contact (from November 1999)*
Tom Pixley, *Netscape Communications Corporation*
Jonathan Robie, *Texcel Research and Software AG*
Peter Sharpe, *SoftQuad Software Inc.*
Chris Wilson, *Microsoft*

[Copyright](#) © 2000 [W3C](#)® ([MIT](#), [INRIA](#), [Keio](#)), All Rights Reserved. W3C [liability](#), [trademark](#), [document use](#) and [software licensing](#) rules apply.

Abstract

This specification defines the Document Object Model Level 2, a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model Level 2 builds on the Document Object Model Level 1.

The DOM Level 2 is made of a set of core interfaces to create and manipulate the structure and contents of a document and a set of optional modules. These modules contain specialized interfaces dedicated to XML, HTML, an abstract view, generic stylesheets, Cascading Style Sheets, Events, traversing the document structure, and a Range object.

Status of this document

This specification is still in the Candidate Recommendation phase. A coordination issue has arisen, which necessitates an extended Candidate Recommendation phase. It will end when the coordination issue is resolved.

Comments on this document are invited and are to be sent to the public mailing list www-dom@w3.org. An archive is available at <http://lists.w3.org/Archives/Public/www-dom/>.

Should this specification prove impossible or very difficult to implement, the necessary changes to make it implementable will be made. If this specification is possible to implement, the only changes which will be made to this specification are minor editorial changes and clarifications.

This document has been produced as part of the [W3C DOM Activity](#). The authors of this document are the DOM WG members. Different modules of the Document Object Model have different editors.

A list of [current W3C Recommendations and other technical documents](#) can be found at <http://www.w3.org/TR>.

Note: The coordination issue affects the handling of namespace URIs. The resolution of the coordination issue may necessitate changes to the DOM Level 2 Core module.

Table of contents

- [Expanded Table of Contents](#)
- [Copyright Notice](#)
- [What is the Document Object Model?](#)
- [Chapter 1: Document Object Model Core](#)
- [Chapter 2: Document Object Model HTML](#)
- [Chapter 3: Document Object Model Views](#)
- [Chapter 4: Document Object Model StyleSheets](#)

- [Chapter 5: Document Object Model CSS](#)
- [Chapter 6: Document Object Model Events](#)
- [Chapter 7: Document Object Model Traversal](#)
- [Chapter 8: Document Object Model Range](#)
- [Appendix A: Changes](#)
- [Appendix B: Accessing code point boundaries](#)
- [Appendix C: IDL Definitions](#)
- [Appendix D: Java Language Binding](#)
- [Appendix E: ECMA Script Language Binding](#)
- [Acknowledgments](#)
- [Glossary](#)
- [References](#)
- [Objects Index](#)
- [Index](#)

[next](#) [contents](#) [objects](#) [index](#)



Leading the Web to its Full Potential...

[Skip to Technologies](#) | [Activities](#) | [Technical Reports](#) | [Site Index](#) | [New Visitors](#) | [About W3C](#) | [Join W3C](#)

The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. W3C is a forum for information, commerce, communication, and collective understanding. On this page, you'll find [W3C news](#), links to [W3C technologies](#) and ways to [get involved](#). New visitors can find help in [Finding Your Way at W3C](#). We encourage you to learn [more about W3C](#).

W3C A to Z

- [Skip to News](#)
- [Accessibility](#)
- [Amaya](#)
- [Annotea](#)
- [CC/PP](#)
- [CSS](#)
- [CSS Validator](#)
- [Device Independence](#)
- [DOM](#)
- [HTML](#)
- [HTML Tidy](#)
- [HTML Validator](#)
- [HTTP](#)
- [Internationalization](#)
- [Jigsaw](#)
- [Libwww](#)
- [MathML](#)
- [Multimodal Interaction](#)
- [Patent Policy](#)
- [PICS](#)

- [PNG](#)
- [Privacy and P3P](#)
- [Quality Assurance \(QA\)](#)
- [RDF](#)
- [Semantic Web](#)
- [SMIL](#)
- [SOAP/XMLP](#)
- [Style](#)
- [SVG](#)
- [TAG](#)
- [URI/URL](#)
- [Voice](#)
- [WAI](#)
- [WebCGM](#)
- [Web Services](#)
- [Web Ontology](#)
- [XForms](#)
- [XHTML](#)
- [XLink](#)
- [XML](#)
- [XML Base](#)
- [XML Encryption](#)
- [XML Key Management](#)
- [XML Query](#)
- [XML Schema](#)
- [XML Signature](#)
- [XPath](#)
- [XPointer](#)
- [XSL and XSLT](#)

[More topics...](#)

News

[Skip to Search](#)

► W3C Co-Hosts XML 2002

4 December 2002: W3C is pleased to co-host [XML 2002](#) to be held 8-13 December in Baltimore, MD, USA. [Chris Lilley](#) participates in a Town Hall panel on the W3C Technical Architecture Group on 10 December. [Philippe Le Hégaré](#) presents W3C Update on 11 December and DOM Level 3 on 12 December. [Daniel Weitzner](#) and [Liam Quin](#), W3C XML Activity Lead, will attend. ([News archive](#))

► EARL 1.0 Working Draft Published

6 December 2002: The Evaluation and Repair Tools Working Group has released the first public Working Draft of the [Evaluation and Report Language \(EARL\) 1.0](#). The specification explains how to use EARL, a general-purpose language for expressing test results, and defines a basic vocabulary. Feedback is welcome. Read about the [Web Accessibility Initiative](#). ([News archive](#))

► W3C Announces Home Page Redesign

5 December 2002: W3C is pleased to announce a home page redesign and accompanying [FAQ](#). Written for newer, standards-compliant user agents in [XHTML 1.0](#) strict, the design features table-less columns and more navigation for [accessibility](#), and [Cascading Style Sheets](#) (CSS) for layout. W3C welcomes your [comments](#). ([News archive](#))

► Multimodal Interaction Use Cases Published

5 December 2002: The Multimodal Interaction Working Group has released [Multimodal Interaction Use Cases](#) as a W3C Note. Airline reservations, driving directions, and name dialing from mobile terminals are analyzed. They highlight device requirements, event handling, network dependencies, and user interaction. Read about the [Multimodal Interaction Activity](#). ([News archive](#))

► Amaya 7.0 Released

3 December 2002: [Amaya](#) is W3C's Web browser and authoring tool. [New features](#) in version 7.0 include user interface enhancements, migration to the Raptor parser, and improved support for XML, SVG, and CSS. [Download Amaya](#) binaries for Solaris, Linux, and Windows. [Source code](#) is available. If you are interested in annotations, visit the [Annotea home page](#). ([News archive](#))

► Speech Synthesis Markup Language Last Call Published

2 December 2002: The Voice Browser Working Group has released a Last Call Working Draft of the [Speech Synthesis Markup Language Version 1.0](#). Comments are welcome through 15 January 2003. With this XML-based language, content authors can generate synthetic speech on the Web, controlling pronunciation, volume, pitch, and rate. Read about the [Voice Browser](#) Activity. ([News archive](#))

► W3C Team Talks in December

2 December 2002: On 3 December, Hugo Haas presents at [Iliatech Club Day on Web Services](#) at INRIA Rocquencourt, Le Chesnay, France, and Charles McCathieNevile presents at [LexiPraxi](#) (in French) at the Agence universitaire de la Francophonie in Paris, France. On 5 December, Kazuhiro Kitagawa gives a keynote at [Internet World Asia](#) in Tokyo, Japan. Several Team members attend [XML 2002](#) in Baltimore, MD, USA held 8-13 December. Daniel Dardailler presents at [Internet: un diritto per tutti](#) (in Italian) in Venice, Italy on 16 December. Browse [upcoming](#) W3C appearances and events. ([News archive](#))

[Past News](#)

Search

[Skip to Contents](#)



[Search W3C Mailing Lists](#)

Contact Us

- [Skip to Footnotes](#)
- [Contact W3C](#)

Get Involved

- [Join W3C](#)
- [Participate](#)
- [Mailing Lists](#)
- [Translations](#)
- [Open Source Software](#)

- [World Offices](#)
- [Employment](#)
- [Subscribe to W3C Weekly News](#)

Mission

- [W3C in Seven Points](#)
- [Frequently Asked Questions](#)
- [Process Document](#)

Member Area

- [Member Home Page](#)
- [Current Members](#)
- [Get Member Password](#)

W3C Team

- [People](#)
- [Past Talks](#)
- [Upcoming](#)

Past News

- [News Archive](#)
- [Press Releases](#)
- [W3C in the Press](#)

Read the [FAQ](#) and [send comments](#) about this page. [Syndicate](#) this page with [RSS 1.0](#), an [RDF](#) vocabulary used for [site summaries](#).

[Webmaster](#) · Last modified: \$Date: 2002/12/06 20:25:40 \$|



[Copyright](#) © 1994-2002 [W3C](#)® ([MIT](#), [INRIA](#), [Keio](#)), All Rights Reserved. W3C [liability](#), [trademark](#), [document use](#) and [software licensing](#) rules apply. Your interactions with this site are in accordance with our [public](#) and [Member](#) privacy statements.



MIT Laboratory for Computer Science

Search

About LCS	who we are, LCS timeline
Research	groups and projects
Publications	technical reports , technical memos
Events / News	calendar, LCS in the news
People	last name beginning with: A-E , F-J , K-O , P-S , T-Z
Resources	jobs available, FAQ , archives
Contact LCS	directions, contact information

LCS Spotlight: [Prof. Daniel Jackson quoted on MIT's OpenCourseware Project in the Chronicle of Higher Education](#)

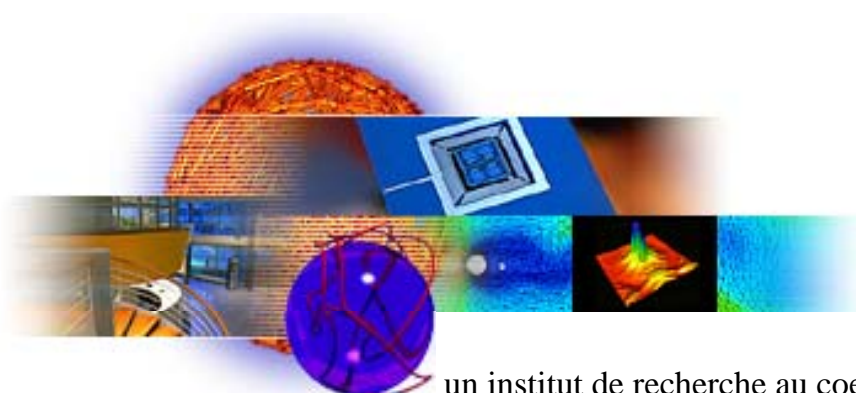
Upcoming Seminar: Dr. David Tennenhouse from Intel Corporation will give a talk titled "Proactive Computing: A Progress Report" on Wednesday December 11th. See the [LCS Event Calendar](#) for more details.

[Research Spotlight](#) CORE - Communication Oriented Routing Environment for Pervasive Computing
Ang-Chih Kao and Larry Rudolph
Pervasive computing environments place a new type of demand of the networking infrastructure. Compared to traditional network, these networks have more nodes (PDA's cellphones, VCR's, et al.), and additionally these nodes have different usage characteristics than traditional nodes in a network. [read more...](#)



MIT Laboratory for Computer Science
200 Technology Square, Cambridge, MA 02139 USA
Tel: 617-253-5851 Fax: 617-258-8682
Email: info@lcs.mit.edu





INSTITUT NATIONAL DE RECHERCHE EN
INFORMATIQUE ET EN AUTOMATIQUE

un institut de recherche au coeur de la société de l'information

L'INRIA recrute

L'INRIA offre **200 emplois** en 2003, du jeune diplômé au spécialiste confirmé, issu du public ou du privé, [chercheurs](#), [ingénieurs et techniciens](#).

13 décembre 2002 : ouverture [concours CR1](#)

Autres [offres](#) en cours (m.a.j. 05/12//02)

A la une

Élections à l'[Académie des sciences](#) :



[Gérard Huet](#) dans la discipline "Informatique"

[Olivier Pironneau](#) dans les disciplines relevant de la



section "Sciences mécaniques"

Spécial presse

Communiqués et dossiers de [presse](#).

27 novembre 2002

ERCIM devient l'hôte européen du **W3C**.

Rapprochement de deux organisations internationales à l'initiative de l'INRIA. [Communiqué](#)

Novembre 2002

Le "**Grid Computing**" à l'INRIA. [Dossier](#)

L'INRIA, un acteur de recherche majeur dans les
Télécommunications

[Dossier](#)

En direct !

[Les Unités de recherche](#) | [Relations internationales](#) |
[INédit](#), lettre d'information | [Rapports d'activité](#) | [Rapports de recherche et thèses](#) | [Logiciels](#) | [Ressources multimédia](#)

- Actualités
- L'INRIA
- Recherche scientifique
- Valorisation & Transfert
- Publications & Documentation
- Travailler et se former à l'INRIA

Siège de l'INRIA ([moyens d'accès](#)) | [Centres de documentation](#)

Domaine de Voluceau

Rocquencourt - B.P. 105

78153 Le Chesnay Cedex - France

Téléphone : +33 1 39 63 55 11

Télécopie : +33 1 39 63 53 30

[Coordonnées des unités](#)

[de recherche et plans d'accès](#)

[A propos du site](#) | [Accès intranet](#) | Webmaster@inria.fr

© INRIA



- 塾長室
- 行事のお知らせ
- 公開講座
- 交通案内・キャンパス案内
- 病院案内
- 図書館案内
- 研究関連
- 採用情報
- 慶應オンライン
- 刊行物
- 情報環境
- 関連組織
- 体育会・スポーツ
- お問い合わせ
- 慶應義塾へのご意見

慶應義塾を知りたい皆様へ



塾生の皆様へ



慶應義塾で学びたい皆様へ



塾員の皆様へ



慶應義塾へのご支援をお考えの皆様へ

最新情報

NEW •y.fX.fef“fh.fO.f%o.fX•z

NYŠw%00'jŽqTfbJ•J•'•d”N,É±,«NY•B•cFnf%00fXj,Â~A“±—D•Ÿ
 ŠefLfff“fpfX“~<G<x<Æ,É”o,ϱ,,m,ç,1
 ‘æ7%00ñ(Ecœä~ãŠw•ÜŽö•ÜŽ@•s,í,ê,é
 ‘æ27%00ñ—•ò•MŽO•Û•~ •¶fRf“fefXfg(E<%0Ê”—•)
 ŽO“c“NŠw%00iŽã•Ã•u%00%00i•u•u< ,Æ“¢<c•1/2•¶|%0»ŽĐ%0iFJfif_ ,ì(E|•p%0Æ,ì•(E»Š““@•|•v(12/12)
 ŽO“c“|_ 12EŽ“†”—, ,’†
 3E(Eϱ<†%0@fvf•fWfFfNfgfZf~fi•[•u’†•,ÌfGflf,çfM•[•EŠÂ<«•EEO•ì(3E•j—â‘è,Æ•j;Eã,Ì“ú’†ŠÖEw•v(12/10•E11)
 •y•m•¶|‘î•Û•z•m’••Ü,ì•â•W
 •m’•fpfŠ•u%00%00iŠJ•Ã
 2003”N“x^ê”Ê“üŽŽ—v•€”—, ,’†
 •V•i•èf^fEf“fLfff“fpfX•E f|•[fvf“fZf~fi•[•Šé<Æ,Æ(Ecœä<•m‘ãŠw,ð,Â,È, @•(12/3•E4•E2003/1/15)
 —•HŠw(Eϱ<†%0Ê,É•æ’[’%0ÊŠw•Z•p••ÛfR•[fX•Ý’u
 •y•m•¶|‘î•Û•zfa f“fgf(Efvf(Efi•[“ü—â•u•À•i11/7•12/12•j
 “ú<gLfLfff“fpfX•@•H,ì(E|•pCfxf“fgfVfŠ•[fY,ì,2“Ä“à•i11/7•`•j
 Ecœä<•m‘ãŠw••ÛŠÖEw%0iŠCŠO”hE—•¶|•â•WŠJŽn
 G-SECfZf~fi•[•ufOf••[fof<ŽŽ‘ã,ÌfAfWfA,Ì(Eo•İ”-“W,Æ’n^æ<|—f•v(12/10)
 ‘æ3%00ñ(Ecœä%0ÊŠw•Z•p•“W(KEIO TECHNO MALL 2002)(12/11)
 fA|[fg•EfZf“f^•[•^%0f%0æ—~ (Eϱ<†%0iFVf“f|fWfEf€•u%0f%0æ,È^,ÆŽŽŠÔ•@,»,ì5•@fW
 ffffbfN•Ef^f••ŠiE€,Ì—ŽãŽã<,Û,1/2,Í“ú•i•¶|Š“,ìfNfŠfefB•[fN•v•i12/17•j
 •u“æ“ñ%00ñ(Ecœä<•m%00%0à“ã%0i(12/21)•v
 Ecœä<•m f|f“fhfŠf“fNf%0fu‘æ169%00ñ“èŠú%00%t%0i(12/23•j
 ‘æ,S,S%00ñ ŽO“c•ÖŠJ•Ã

キャンパス

- @•@•EŽO“c
- @•@•E“ú<g
- @•@•E—î•ã
- @•@•E•Ã“î“„ò
- @•@•E•M”Z’¬

一貫教育・学部・大学院

- @•fēšÑ•ç„
- @•E—c'ZĒ
- @•E•Ê”
- @•E†“TM.”
- @•E.Ä“‘‘‘ò’+E.“TM.”
- @•E.“TMŠwZ
- @•EŽŭ–Ø.“TMŠwZ
- @•E.—žq.“TMŠwZ
- @•E.m.xŠw%@•j.“TM.”;j

[illegible]

- @•f•ăŠw%o@•,•
- @•@•EĖO%oċŠĈ—•EĖ<†%oĚ
- @•@•@•@•@•f†fWf†fXfXfN•|f•|
- @•@•E•ŸŠwEĖ<†%oĚ
- @•@•EĖO•ŸŠwEĖ<†%oĚ
- @•@•E—ŠwEĖ<†%oĚ
- @•@•EŽĐ%oĩŠwEĖ<†%oĚ
- @•@•E•ĖŠwEĖ<†%oĚ
- @•@•EăŠwEĖ<†%oĚ
- @•@•E—HŠwEĖ<†%oĚ
- @•@•E••ô•Ef•ffBfAĖĖ<†%oĚ

•E-@%È•ê-â‘ăŠw%o@•i%o¼•İ•j•iŠJ•Ý•€”õ’†•

- i [‘æ31Šú‘æ1%õñ•\]c^õ%ĩŠJ•Ā](#)
 - i [ĒEcœä<•m‘ăŠw•AAPRU,É%ŌĀ–i](#)
 - i [ĒEcœä<•m‘ăŠw•A,q,k,f,É%ŌĀ–i](#)
 - i [21•¢ICOEfvf•fOf%Ōf€•@5•ª–ì,.,×,Ā,É‘T’è,³,ê,é](#)
 - i [•½•–15”N“‘x•,,‘E“üŽŽ•F•,“™Šw•Z•.,‘E“üŽŽ•EŽu–Ø•,“™Šw•ZŽ©ĒÈ•,‘E“üŽŽ](#)
 - i [‘æ7%õñĒEcœä^ăŠw•ŬŽó•ŬŽÒĒ~`è](#)
 - i [•V‘ăŠw%Ō@•‘z•iPDFftf@fCf•j•F–@%ŌĒ‘ăŠw%Ō@•i%Ō¼•İ•j•Eİ–ª•\‘z‘ăŠw%Ō@•i%Ō¼•İ•j•EĒŌ%Ōc‘ăŠw%Ō@•i%Ō¼•İ•j](#)
- [>>More](#)

•@•f’ZŠú‘ăŠw•E•’Šw•Z•,,
•@•@•EŠĀĒi’ZŠú‘ăŠw
•@•@•EŠŌ•‘ĒêŠw•Z