Introduction to XPath in Java using Jaxen

SD West 2003

Bob McWhirter

The Werken Company

bob@werken.com

March 28, 2003

http://jaxen.org/

Quickest Intro

- Open-Source.
- Business-friendly license (ASF/BSD).
- Works with most XML object-models.

Open-Source

Being open-source, jaxen has numerous developers working on various aspects of the project, from supporting additional object-models to seeking out optimization opportunities.

Jaxen has existed for about two years, and has over a dozen committers, with 2-to-3 active at any point in time. Some committers are simply users of jaxen, others have created custom model adapters, while still others are implementors of other open-source XML object-models.

Business-friendly License

Since jaxen uses a license similar to the one used by the Apache Software Foundation, there are few restrictions on its usage. It may be used in other open-source projects or in closed-source commercial products. The only requirement is that the code maintains its copyrights.

That's it!

Works with Most XML Object-Models

- **dom4j** James Strachan's dom4j includes direct support for XPath by using the Jaxen library.
- **JDOM** Jason Hunter's JDOM includes optional support for XPath by using the Jaxen library.
- **W3C DOM** Jaxen supports DOM documents.
- **EXML** The Mind Electric's EXML 6.0 includes direct support for XPath by using the Jaxen library.

History

The first XPath engine Bob created was werken.xpath, and it worked only with JDOM. It was based upon an ANTLR parser-generator grammar and contained numerous bugs. James Strachan started the dom4j project. Initially, we worked with porting werken.xpath to dom4j, but maintaining separate codebases proved difficult.

And thus, the concept for jaxen was born. . .

(History - The Beginnings)

Due to issues with the ANTLR-based grammar, SAXPath, a hand-rolled expression lexer and parser was written. SAXPath parses and reports XPath expressions in a manner similar to how SAX works for XML content.

Bob McWhirter and James Strachan designed the Navigator object-model adapter and implemented the core engine. James wrote the binding to dom4j while Bob wrote the bindings for JDOM and EXML. James Strachan and David Megginson created the W3C DOM binding.

(History - Contributors)

Many others contributed patches, optimizations and improvements:

- Erwin Boldwidt
- Eddie McGreal
- Jan Dvorak
- Mark A. Belonga
- Michael Brennan
- Stephen Colebourne
- Paul R. Brown
- Alex Chaffee
- Steen Lehmann
- Attila Szegedi
- Christian Nentwich
- Pete Kazmier
- Jeffrey Brekke

Jaxen and the XML-InfoSet

jaxen's flexibility comes from the fact that it works purely in terms of the XML InfoSet instead of any concrete XML representation. This flexibility actually allows jaxen to work with non-XML models. The only requirement is that a Navigator adapter be written to satisfy the subset of the InfoSet required by jaxen.

XML-InfoSet

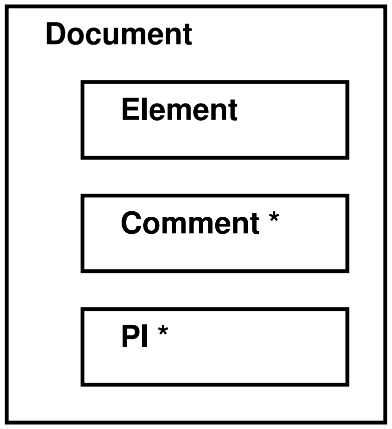
The XML-InfoSet defines the semantics of XML instead of the textual serialization as the original XML 1.0 spec does. The InfoSet recognizes that the true structure is a tree with various types of nodes which the spec refers to as “information items”.

- Document Information Item
- Element Information Items
- Attribute Information Items
- Processing Instruction Information Items
- Character Information Items
- Comment Information Items
- Namespace Information Items

(InfoSet - Document Info Item)

The Document information item is typically modeled in XML object-model frameworks by an explicit class such as org.dom4j.Document or org.jdom.Document.

The document information item most importantly contains children, including the root element and any comments or processing-instructions outside of the root element.



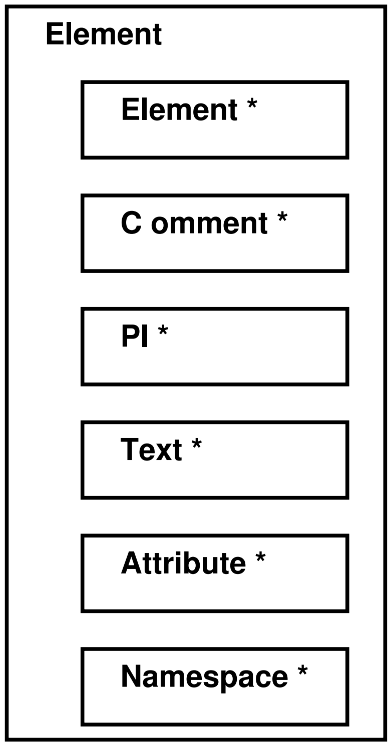
(InfoSet - Element Info Items)

Each tag in an XML document is an element and is represented by the Info-Set as an Element information item. Once again, most XML object-model frameworks represent each element with an instance of an element class, such as org.dom4j.Element or org.jdom.Element.

An element information item contains several types of children, including attributes, comments, namespace declarations, text context and other elements. It also contains information regarding its own name and namespace.

```
<article id="mcw03"
         xmlns="http://jaxen.org/example-ns/">
```

(InfoSet - Element Info Items - Diagram)



(InfoSet - Attribute Info Items)

Each attribute that does not begin with xmlns is represented by an Attribute information item. Most object-models represent them with an explicit attribute class such as org.dom4j.Attribute or org.jdom.Attribute. It contains information regarding its own name and namespace.

```
<article id="mcw03"
         xmlns="http://jaxen.org/example-ns/">
```

(InfoSet - Processing-Instruction Info Items)

Processing-instructions, while not widely used, embody their target and the text associated with them. Different object-models represent PIs differently, so no generalized statement may be made.

```
<?mycompany:insert-random-bugs true?>
```

(InfoSet - Character Info Items)

The InfoSet defines a Character information item for each individual text character. The Navigator of jaxen works with consecutive spans of uninterrupted characters.

```
<text>
    jaxen is a fun and exciting way to drive your coworkers insane..
</text>
```

(InfoSet - Comment Info Items)

Comment information items appear as children of either Document or Element items and contain the textual content of the comments themselves.

```
<!--
  || Comment on the Document information item
  -->
<journal xmlns="http://jaxen.org/example-ns/">
    <!--
      || Comment on the Element information item for <journal>
      -->
</journal>
```

NOTE: Due to how the specifications are written, there is no guarantee that XML parsed from a file will ever contain comment items. Parsers are allowed to discard the comments and so they may not be included in your model of choice.

(InfoSet - Namespace Info Items)

The Namespace information items contained by Element items represent all XML namespace bindings in effect at the scope of the element.

```
<journal xmlns="http://jaxen.org/example-ns-1/">
    <art:article xmlns:art="http://jaxen.org/example-ns-2">
```

What is XPath?

XPath is a node-addressing expression language for the XML InfoSet.

XPath expressions are used to traverse the graph provided by the InfoSet in order to locate any node contained therein.

- Full expression language.
- Multiple 'directions' of traversal.
- Predicate evaluation for filtering.
- Extremely extensible.

Compared to XQuery

XPath and XQuery vaguely overlap in functionality. XPath 2.0 overlaps even more so with XQuery.

In general, XQuery is more rigorously defined, more type-safe, much larger, and not finished.

XPath is strictly an addressing language, not a full query language, but most people find that it satisfies the 80/20 rule where it provides 80% of the solution for 20% of the effort.

Simple Expressions

Math!

Since XPath is a full expression language, arbitrary arithmetic constitutes valid expressions.

```
42 + 84.2
```

```
10 div 3
```

```
(1 + 3) * 42
```

Location Paths

Location paths are the core of the XPath expression language with regards to XML documents. A location path is comprised of a series of steps.

Each step is evaluated, in order, against the results of the previous step. The result of each step is a possibly empty set of some nodes from the document.

```
/journal/article/author
```

What that means. . .

(Location Paths - Processing Logic)

An XPath is evaluated in relation to some initial context which typically is a Document node from an object-model.

Examples of common initial context classes:

- org.dom4j.Document
- org.jdom.Document
- org.w3c.dom.Document

(Location Paths - Processing Logic - Example Document)

Given a document with the structure:

```
<journal>
    <article id="article.1">
        <title>...</title>
        <author>
          <first>Bob</first>
          <last>McWhirter</last>
        </author>
        <text>
        </text>
    </article>
    <article id="article.2">
        <title>...</title>
        <author>
          <first>James</first>
          <last>Strachan</last>
        </author>
        <text>
        </text>
    </article>
</journal>
```

(Location Paths - Processing Logic - Example XPath)

Given an XPath expression:

```
/journal/article/author/last
```

Let's walk through how it is evaluated to select all `<last>` elements which are children of `<author>` elements which in turn are children of `<article>` elements that have a parent `<journal>` element that is the root element of a document.

. . . whew. . .

The initial slash character ("/") indicates that regardless of the initial context, the path is an absolute location path and thus starts at the very top of the document.

(Location Paths - Processing Logic - Journal)

/journal/article/author/last

```
<journal>
    <article id="article.1">
        <title>...</title>
        <author>
          <first>Bob</first>
          <last>McWhirter</last>
        </author>
        <text>
        </text>
    </article>
    <article id="article.2">
        <title>...</title>
        <author>
          <first>James</first>
          <last>Strachan</last>
        </author>
        <text>
        </text>
    </article>
</journal>
```

(Location Paths - Processing Logic - Article)

/journal/article/author/last

```
<journal>
    <article id="article.1">
        <title>...</title>
        <author>
          <first>Bob</first>
          <last>McWhirter</last>
        </author>
        <text>
        </text>
    </article>
    <article id="article.2">
        <title>...</title>
        <author>
          <first>James</first>
          <last>Strachan</last>
        </author>
        <text>
        </text>
    </article>
</journal>
```

(Location Paths - Processing Logic - Author)

`/journal/article/author/last`

```
<journal>
    <article id="article.1">
        <title>...</title>
        <author>
          <first>Bob</first>
          <last>McWhirter</last>
        </author>
        <text>
        </text>
    </article>
    <article id="article.2">
        <title>...</title>
        <author>
          <first>James</first>
          <last>Strachan</last>
        </author>
        <text>
        </text>
    </article>
</journal>
```

(Location Paths - Processing Logic - Last)

`/journal/article/author/last`

```
<journal>
    <article id="article.1">
        <title>...</title>
        <author>
          <first>Bob</first>
          <last>McWhirter</last>
        </author>
        <text>
        </text>
    </article>
    <article id="article.2">
        <title>...</title>
        <author>
          <first>James</first>
          <last>Strachan</last>
        /author>
        <text>
        </text>
    </article>
</journal>
```

(Location Paths - Processing Logic - Results)

The result of evaluating the XPath against the document is a node-set that contains nodes directly from the original object-model. The results are not copies.

In this case, the results are two element nodes, being instances of classes such as:

- org.dom4j.Element
- org.jdom.Element
- org.w3c.dom.Element

(Location Paths - Attributes)

The previous example demonstrated some of the simplest location path expressions possible. Not only can a location path select elements, they can also select attributes amongst other items.

A step that begins with the “`@`” character selects an attribute with the given name instead of an element.

(Location Paths - Attributes)

`/journal/article/@id`

This path would select the `id` attribute node from each of the `<article>` tags. Once again, it selects instances of the actual corresponding classes for attributes in the target object-model, such as:

- **org.dom4j.Attribute**
- **org.jdom.Attribute**
- **org.w3c.dom.Attr**

The path does not select the values of the attributes.

(Location Paths - Attributes - Example)

`/journal/article/@id`

```
<journal>
    <article id="article.1">
        <title>...</title>
        <author>
          <first>Bob</first>
          <last>McWhirter</last>
        </author>
        <text>
        </text>
    </article>
    <article id="article.2">
        <title>...</title>
        <author>
          <first>James</first>
          <last>Strachan</last>
        </author>
        <text>
        </text>
    </article>
</journal>
```

(Location Paths - Predicates)

Sometimes it is desirable to narrow the results of a particular step of a location path. This narrowing is the job of a predicate. Multiple predicates can be chained together to further and further constrain the result node set.

```
/journal/article[@id='article.2']
```

After a step has been evaluated, if a predicate follows, then the predicate is evaluated in relation to each member of the node set.

(Location Paths - Predicates - Base)

`/journal/article[@id='article.2']`

```
<journal>
    <article id="article.1">
        <title>...</title>
        <author>
          <first>Bob</first>
          <last>McWhirter</last>
        </author>
        <text>
        </text>
    </article>
    <article id="article.2">
        <title>...</title>
        <author>
          <first>James</first>
          <last>Strachan</last>
        </author>
        <text>
        </text>
    </article>
</journal>
```

(Location Paths - Predicates - 1st article)

```
/journal/article[@id='article.2']
```

```
<journal>
    <article id="article.1">
        <title>...</title>
        <author>
          <first>Bob</first>
          <last>McWhirter</last>
        </author>
        <text>
        </text>
    </article>
    <article id="article.2">
        <title>...</title>
        <author>
          <first>James</first>
          <last>Strachan</last>
        </author>
        <text>
        </text>
    </article>
</journal>
```

(Location Paths - Predicates - 2nd article)

```
/journal/article[@id='article.2']
```

```
<journal>
    <article id="article.1">
        <title>...</title>
        <author>
          <first>Bob</first>
          <last>McWhirter</last>
        </author>
        <text>
        </text>
    </article>
    <article id="article.2">
        <title>...</title>
        <author>
          <first>James</first>
          <last>Strachan</last>
        </author>
        <text>
        </text>
    </article>
</journal>
```

(Location Paths - Predicates - Proximity)

The previous predicate was based upon some content as opposed to position within the document. XPath defines the concept of a proximity predicate that allows selection of specific elements by their location using roughly array notation.

```
/journal/article[2]
```

Selects the second `<article>` under the `<journal>` tag. Since proximity predicates rely only on the positions of nodes, they are fragile and do not survive large-scale editing of the document.

NOTE: Unlike Java, XPath indices begin at `1` instead of `0`.

(Location Paths - Predicates - Proximity)

`/journal/article[2]`

```
<journal>
    <article id="article.1">
        <title>...</title>
        <author>
          <first>Bob</first>
          <last>McWhirter</last>
        </author>
        <text>
        </text>
    </article>
    <article id="article.2">
        <title>...</title>
        <author>
          <first>James</first>
          <last>Strachan</last>
        </author>
        <text>
        </text>
    </article>
</journal>
```

(Location Paths - Predicates - Multiple)

Predicates can be chained together to recursively refine the selection.

```
/journal/article[author/last='Strachan'][1]
```

This path would only select the first `<article>` that was written by an author with the last name `Strachan`.

(Location Paths - Predicates - Nested)

Predicates contain any arbitrary XPath expression which can include other location paths and predicates.

```
/journal/article[author[1]/last='Strachan']
```

This path would select all `<article>` elements whose first `<author>` has the `<last>` name of `Strachan`.

(Location Paths - Predicates - Nested - Example)

```
/journal/article[author[1]/last='Strachan']
```

```
<journal>
    <article>
        <title>...</title>
        <author>
          <first>James</first>
          <last>Strachan</last>
        </author>
        <author>
          <first>Bob</first>
          <last>McWhirter</last>
        </author>
        <text>
        </text>
    </article>
</journal>
```

Jaxen API

Model Bindings

Several object-models, such as dom4j and EXML include bindings to the jaxen xpath engine through their own APIs, typically in the form of selectNodes(String xpathExpr) or selectSingleNode(String xpathExpr) methods on their Document and Element classes.

Here, we'll address using jaxen directly in conjunction with an object-model. Please consult your chosen model's API documentation for any native bindings it may have.

The XPath Classes

The main interface for working with jaxen XPaths is org.jaxen.XPath.

```
package org.jaxen;

public interface XPath
{
    Object evaluate(Object context)
        throws JaxenException;

    ...
}
```

Each supported object-model has a matching concrete implementation:

- **org.jaxen.dom4j.Dom4jXPath**
- **org.jaxen.jdom.JDOMXPath**
- **org.jaxen.dom.DOMXPath**

(The XPath Classes - Constructing)

Once the appropriate class has been selected and imported into your code, a new instance of the XPath class may be created using the constructor that takes a string XPath expression.

```
try
{
    XPath xpath = new Dom4jXPath( "/journal/article/[author/last='Strachan']" );
}
catch (JaxenException e)
{
    e.printStackTrace();
}
```

(The XPath Classes - Evaluation)

Once an XPath has been instantiated successfully, it may be evaluated against multiple different contexts. The XPath implementations are thread-safe and so may be cached and shared by multiple threads.

```
try
{
    XPath xpath = new Dom4jXPath( "/journal/article/[author/last='Strachan']" );

    Document doc = retrieveDocument();

    List results = (List) xpath.evaluate( doc );
}
catch (JaxenException e)
{
    e.printStackTrace();
}
```

(The XPath Classes - Evaluation - Results)

As noted before, while the XPath is primarily intended for manipulating XML, it is a full expression language capable of evaluating non-XML-based expressions. The return value of evaluate(...) is a java.lang.Object.

Calling code must cast the result to the appropriate class. When working with location paths, the result will always be a java.util.List which represents the possibly empty node set of selected nodes.

The members of the List will be instances of classes from the underlying object-model. In other cases, it may return java.lang.Number, java.lang.Boolean, or java.lang.String.

(The XPath Classes - Evaluation - Helpers)

The XPath interface allows for additional helper methods that handle much of the desired conversion and casting.

```
package org.jaxen;

public interface XPath
{
    String stringValueOf(Object context)
        throws JaxenException;

    boolean booleanValueOf(Object context)
        throws JaxenException;

    boolean numberValueOf(Object context)
        throws JaxenException;

    List selectNodes(Object context)
        throws JaxenException;

    Object selectSingleNode(Object context)
        throws JaxenException;
}
```

(The XPath Classes - Evaluation - stringValueOf)

The stringValueOf(...) method performs a normal evaluate(...) and then follows XPath's rules for coercion of the result to a string.

In terms of XPath, this means that the first node is converted to its string value and all others are discarded.

```
/journal/article/author/last
```

While this path would select both of the `<author>` tags, only the first would be converted to its string value, which for elements, is the complete text content.

This would result in the string `McWhirter`.

(The XPath Classes - Evaluation - booleanValueOf)

The booleanValueOf(...) method performs a normal evaluate(...) and then follows XPath's rules for coercion of the result to a boolean.

An empty result set is interpreted as `false` while a non-empty result set is interpreted as `true`.

This XPath would return `true`:

```
/journal/article/author[last='McWhirter']
```

This XPath would return `false`:

```
/journal/article/author[last='MacWithier']
```

(The XPath Classes - Evaluation - numberValueOf)

The numberValueOf(...) method performs a normal evaluate(...) and then follows XPath's rules for coercion of the result to a number.

In terms of XPath, this means that the first node is converted to its number value and all others are discarded.

(The XPath Classes - Evaluation - selectNodes)

The selectNodes(...) method makes sense only for XPaths that perform node selection and not arithmetic. It simplifies calling code by performing the cast to a List on the results of evaluate(...).

This XPath would return all `<author>` elements in a list.

`/journal/article/author`

```
try
{
    List authors = xpath.selectNodes( "/journal/article/author" );
}
catch (JaxenException e)
{
    e.printStackTrace();
}
```

(The XPath Classes - Evaluation - selectSingleNode)

The selectSingleNodes(...) operates similar to selectNodes(...) but only returns the first members of the selected list as an Object. Depending on the expression, calling code must cast to a class appropriate to the object-model being used.

This XPath would return only the first `<author>` element.

```
/journal/article/author
```

```
try
{
    Element firstAuthor = (Element) xpath.selectSingleNode( "/journal/article/author" );
}
catch (JaxenException e)
{
    e.printStackTrace();
}
```

Advanced XPath & Jaxen

Namespaces - Variables - Functions

Namespaces

In many modern XML documents, XML Namespaces are used to help differentiate tags. Multiple tags may have the same name but exist in different namespaces. These are considered unique.

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <soap:Body>
       <mail:Envelope
         xmlns:mail="http://jaxen.org/example-ns/mail/">
         <mail:to>bob@werken.com</mail:to>
       </mail:Envelope>
   </soap:Body>
</soap:Envelope>
```

Here, two different `<Envelope>` tags exist.

One exists in the **http://schemas.xmlsoap.org/soap/envelope** namespace while the other is in the **http://jaxen.org/example-ns/mail/** namespace.

(Namespaces - URIs)

The namespace of an element is a URI that does not necessarily have to be resolvable or point to any particular type of resource. It acts purely as a distinguishing identifier.

It would be unwieldy to affix the namespace URI to each element, so XML defines a way to declare a prefix mapping for each namespace.

This is accomplished by adding a pseudo-attribute to an element.

```
<wj:journal xmlns:wj="http://werken.com/werken-journal/">
  <wj:article>
    ...
  </wj:article>
</wj:journal>
```

Any pseudo-attribute that begins with xmlns is considered to be a namespace prefix mapping within the scope of the element upon which it is defined.

(Namespaces - Prefix Mapping)

The normal form of a namespace prefix mapping declaration is:

```
xmlns:${PREFIX}="${URI}"
```

```
<tagname xmlns:myprefix="my-namespace">
```

The declaration is available for the tag upon which it is declared and any nested child tag. A prefix mapping is not required to be used by the tag upon which it is defined, and multiple mappings may be declared upon a single tag.

```
<a:tagname xmlns:a="uri-a" xmlns:b="uri-b">
```

(Namespaces - Default Mapping)

A default namespace mapping can be used to define which namespace tags are a part of, unless otherwise specified. The format of the default mapping is identical to the prefix mapping, with the exception that a prefix is not used.

```
xmlns="${URI}"
```

```
<tagname xmlns="my-namespace">
```

In this case, `<tagname>` has no prefix but a default namespace mapping is defined, so `<tagname>` is a member of `my-namespace`.

(Namespaces - Prefixes Do Not Matter)

The actual prefixes used within a document **do not matter**. Only the mapped namespace URI is important. These three documents are semantically identical:

Using the default namespace mapping functionality:

```
<journal xmlns="http://werken.com/werken-journal/">
</journal>
```

Using the prefix namespace mapping functionality:

```
<yak:journal yak:xmlns="http://werken.com/werken-journal/">
</yak:journal>
```

Using the prefix namespace mapping functionality with a different prefix:

```
<frobnovich:journal frobnovich:xmlns="http://werken.com/werken-journal/">
</frobnovich:journal>
```

(Namespaces - Prefixes - XPath)

Since prefixes do not matter, how do you construct an XPath that works with namespaces?

Using prefixes!

Just as each element in a document has a set of prefix-to-namespace mappings, an XPath expression may also contain a set of prefix-to-namespace mappings.

The only caveat is that XPath has absolutely no concept of a default namespace mapping.

```
j:journal/j:article/j:author/j:last
```

(Namespaces - Prefixes - XPath - Mappings)

The XPath specification does not address how prefix namespace mappings are created. It only specifies that an XPath is evaluated within the scope of a namespace context which defines the mappings.

In XSLT, the namespace context is composed of all namespace mappings in effect within the template (not the target document) at the point the xpath is used.

```
<xsl:template match="wj:author"
              xmlns:wj="http://werken.com/werken-journal/">
    ...
</xsl:template>
```

(Namespaces - Prefixes - XPath - Mappings)

The template will match `<author>` tags in the http://werken.com/werken-journal/ namespace, regardless of the actual prefix (or default mapping) used within the target document.

```
<journal xmlns="http://werken.com/werken-journal/">
    <author>
        ...
    </author>
</journal>

<yak:journal xmlns:yak="http://werken.com/werken-journal/">
    <yak:author>
        ...
    </yak:author>
</yak:journal>
```

(Namespaces - Jaxen)

Since jaxen is purely an XPath engine, and not an XSLT engine, it follows the specification in mandating nothing about how namespace prefix mappings are generated, but simply a namespace context is available.

This is accomplished through the **org.jaxen.NamespaceContext** interface. It contains but a single method declaration for translating a prefix to a namespace URI.

```
package org.jaxen;

public interface NamespaceContext
{
    String translateNamespacePrefixToUri(String prefix);
}
```

(Namespaces - Jaxen - SimpleNamespaceContext)

Since mapping a prefix to a namespace URI is a perfect job for a look-up table, jaxen provides the **org.jaxen.SimpleNamespaceContext**, which is an implementation simply backed by a hash-map.

```
package org.jaxen;

public class SimpleNamespaceContext
    implements NamespaceContext
{
    public SimpleNamespaceContext() { ... }

    public SimpleNamespaceContext(Map namespaces) { ... }

    public void addNamespace(String prefix,
                             String namespaceUri) { ... }

    public String translateNamespacePrefixToUri(String prefix) { ... }
}
```

(Namespaces - Jaxen - Using NamespaceContext)

Each XPath has a NamespaceContext associated with it. The association is made using the setNamespaceContext(...) method.

```
package org.jaxen;

public interface XPath
{
    ...
    void setNamespaceContext(NamespaceContext namespaceContext);
    ...
}
```

Any prefix that is resolvable through the NamespaceContext may be used within the XPath expression itself. If code using the XPath does not explicitly set a NamespaceContext then a default context that contains no mappings is used.

(Namespaces - Jaxen - Example)

```
try
{
    XPath xpath = new Dom4jXPath( "/j:journal/j:article/j:author" );

    SimpleNamespaceContext nsContext = new SimpleNamespaceContext();

    nsContext.addNamespace( "j"
                            "http://werken.com/werken-journal/" );

    xpath.setNamespaceContext( nsContext );

    Document journalDoc = getJournalEdition( 42 );

    List authors = xpath.selectNodes( journalDoc );
}
catch (JaxenException e)
{
    e.printStackTrace();
}
```

Variables

The XPath specification allows for variables in expressions to allow parameterization at evaluation time.

Similar to the namespace-context, each XPath expression also has a variable-context that maps variable names to values.

```
/journal/article/author[last=$lastName]
```

Or with namespace support:

```
/journal/article/author[last=$myNsPrefix:lastName]
```

(Variables - Jaxen - VariableContext)

Parallel to the NamespaceContext, jaxen provides an interface VariableContext and a useful simple implementation.

```
package org.jaxen;

public interface VariableContext
{
    Object getVariableValue(String namespaceUri,
                            String prefix,
                            String localName)
        throws UnresolvableException;
}
```

The three parameters to the getVariableValue(...) method are:

1. **namespaceUri** The namespace URI associated with the prefix as determined by the current NamespaceContext.
2. **prefix** The actual prefix used in the XPath expression.
3. **localName** The portion of the variable name that is not the namespace prefix.

(Variables - Jaxen - SimpleVariableContext)

A simple implementation of VariableContext is provided by the sensibly-named SimpleVariableContext, which is backed by a hash-map.

```
package org.jaxen;

public class SimpleVariableContext
    implements VariableContext
{
    public SimpleVariableContext() { ... }

    public void setVariableValue(String namespaceUri,
                                 String localName,
                                 Object value) { ... }

    public void setVariableValue(String localName,
                                 Object value) { ... }

    public Object getVariableValue(String namespaceUri,
                                   String prefix,
                                   String localName)
        throws UnresolvableException { ... }
}
```

(Variables - Jaxen - Using VariableContext)

Each XPath has a VariableContext associated with it. The association is made using the setVariableContext(...) method.

```
package org.jaxen;

public interface XPath
{
    ...
    void setVariableContext(VariableContext variableContext);
    ...
}
```

If code using the XPath does not explicitly set a VariableContex then a default context that contains no variables is used.

(Variables - Jaxen - Example)

```
try
{
    XPath xpath = new Dom4jXPath( "/journal/article[author/last=$lastName]" );

    SimpleVariableContext varContext = new SimpleVariableContext();

    varContext.setVariable( "lastName"
                            "Strachan" );

    xpath.setVariableContext( varContext );

    Document journalDoc = getJournalEdition( 42 );

    List strachanArticles = xpath.selectNodes( journalDoc );
}
catch (JaxenException e)
{
    e.printStackTrace();
}
```

Functions

The XPath language supports functions in expressions and provides for a core library of functions dealing with strings, numbers, booleans and node sets.

Determine the number of articles written by Mr. Strachan:

```
count(/journal/article[author/last="Strachan"])
```

Find the Irish:

```
/journal/article[starts-with(author/last,"Mc")]
```

(Functions - Core Library)

The core XPath function library is divided into four groups:

1. **Node set functions**
   Functions for working with node-sets. Either the implicit current node set or one passed as a parameter.

2. **String functions**
   Functions for working with strings. Includes type coercions.

3. **Boolean functions**
   Functions for working with booleans. Includes type coercions.

4. **Number functions**
   Functions for working with numbers. Includes type coercions.

(Functions - Core Library - Node Set Functions)

- **last()** Returns the index of the last item of the current result set.

```
/journal/article[last()]
```

- **position()** Returns the index of the current item in the current result set.

```
/journal/article[position()<3]
```

- **count(*node-set*)** Returns the number of items in the parameter result set.

```
count(/journal/article)
```

- **id(*object*)** Returns the elements with the ID specified.

```
id("article.1")/author/last
```

(Functions - Core Library - Node Set Functions)

- **local-name(*node-set?*)** Returns the non-namespace portion of the node name of either a node set passed as a parameter or the current node in the current node set.

  ```
  local-name(/wj:journal)

  /journal/*[local-name()="article"]
  ```

- **namespace-uri(*node-set?*)** Returns the namespace URI of the node name of either a node set passed as a parameter or the current node in the current node set.

  ```
  namespace-uri(/wj:journal)

  /journal/*:*[namespace-uri()="http://werken.com/werken-journal/"]
  ```

- **name(*node-set?*)** Returns the complete textual node name of either a node set passed as a parameter or the current node in the current node set.

  ```
  name(/journal)

  /*[name()="soap:Envelope"]
  ```

(Functions - Core Library - String Functions)

- **string(*object?*)** Converts an object (possibly the current context node) to its string value.

```
/journal/article/author[string()='Strachan']
```

- **concat(*string, string, string**)** Concatenate two or more strings together.

```
concat(author/salutation, ' ', author/last)
```

- **starts-with(*string, string*)** Determine if the first argument starts with the second argument string.

```
/journal/article[starts-with(title, 'Advanced')]
```

- **contains(*string, string*)** Determine if the first argument contains the second argument string.

```
/journal/article[contains(title, 'XPath')]
```

(Functions - Core Library - String Functions)

- **substring-before(*string*, *string*)** Retrieve the substring of the first argument that occurs before the first occurrence of the second argument string.

```
substring-before(/journal/article[1]/date, '/')
```

- **substring-after(*string*, *string*)** Retrieve the substring of the first argument that occurs after the first occurrence of the second argument string.

```
substring-after(/journal/article[1]/date, '/')
```

- **substring(*string*, *number*, *number?*)** Retrieve the substring of the first argument starting at the index of the second number argument, for the length of the optional third argument.

```
substring('McStrachan', 3)
```

- **string-length(*string?*)** Determine the length of a string, or the current context node coerced to a string.

```
/journal/article[string-length(author/last) > 9]
```

(Functions - Core Library - String Functions)

- **normalize-space(*string?*)** Retrieve the string argument or context node with all space normalized, trimming whitespace from the ends and compressing consecutive whitespace elements to a single space.

```
normalize-space(/journal/article[1]/content)
```

- **translate(*string*, *string*, *string*)** Retrieve the first string argument augmented so that characters that occur in the second string argument are replaced by the character from the third argument in the same position.

```
translate( 'bob', 'abc', 'ZXY' ) XoX
```

(Functions - Core Library - Boolean Functions)

- **boolean(*object*)** Convert the argument to a boolean value.

```
boolean(/journal/article/author/last[.='Strachan']
```

- **not(*boolean*)** Negate a boolean value.

```
not(/journal/article/author/last[.='Strachan']
```

- **true()** Boolean true.
- **false()** Boolean false.
- **lang(*string*)** Test if the lang, as set by `xml:lang` attributes is the language specified.

```
/journal/article[1]/content[lang('en')]
```

(Functions - Core Library - Number Functions)

- **number(*object?*)** Convert the argument or context node to a number value.

```
/journal[number(year)=2003]
```

- **sum(*node-set*)** Sum the values of the node-set.

```
sum(/journal/article/author/age)
```

- **floor(*number*)** Return the largest integer that is not greater than the number argument.
- **ceiling(*number*)** Return the smallest integer that is not less than the number argument.
- **round(*number*)** Round the number argument.

(Functions - Function Context)

Like most other things in XPath, there is a function context that contains the core library of functions. The set of functions available within an expression is extensible.

XSLT has added the document(*url*) function. Other technologies, such as XPointer and BPEL4WS add even more functions to the XPath function context.

(Functions - Jaxen - Function Context)

An extensible function context is supported in jaxen through the FunctionContext interface.

```
package org.jaxen;

public interface FunctionContext
{
    Function getFunction(String namepsaceUri,
                         String prefix,
                         String localName)
        throws UnresolvableException;
}
```

The three parameters to the getFunction(...) method are:

1. **namespaceUri** The namespace URI associated with the prefix as determined by the current NamespaceContext.
2. **prefix** The actual prefix used in the XPath expression.
3. **localName** The portion of the variable name that is not the namespace prefix.

Quick Romp Through Advanced XPath

Advanced Axes

XPath provides many different semantic methods for navigating a document. Each direction is an axis that defines which nodes each step will be applied to. Some we have already visited.

Each axis allows the effects of a particular step to be constrained to a certain set of nodes for matching.

The general syntax for a step with an explicit axis is:

```
${AXIS}::${NAME}
```

```
/child::journal/child::article/attribute::id
```

```
/journal/article/@id
```

(Advanced Axes - Descriptions)

- **child** Children of the context node. This is the default implicit axis.
- **descendant** Descendent of the context node.
- **parent** Parent of the context node.
- **ancestor** Ancestors of the context node.
- **following-sibling** Following siblings of the context node.
- **preceding-sibling** Preceding siblings of the context node.
- **following** Nodes following the context node.
- **preceding** Nodes preceding the context node.
- **attribute** Attributes of the context node. Steps begining with '`@`' operate along the attribute axis.
- **namespace** Namespaces of the context node.
- **self** The context node.
- **descendant-or-self** The context node or its descendants.
- **ancestor-or-self** The context node or its ancestors

(Advanced Axes - Examples)

In the following slides, the context node will be highlighed in blue and the axis will be demonstrated in red. When the context node is a part of the axis, it will be highlighted in purple.

(Advanced Axes - Examples - child)

```
<library>
    <journal> ... </journal>
    <journal>
        <article id="article.1">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.2">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.3">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
    </journal>
    <journal> ... </journal>
</library>
```

(Advanced Axes - Examples - descendant)

```
<library>
    <journal> ... </journal>
    <journal>
        <article id="article.1">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.2">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.3">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
    </journal>
    <journal> ... </journal>
</library>
```

(Advanced Axes - Examples - parent)

```
<library>
    <journal> ... </journal>
    <journal>
        <article id="article.1">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.2">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.3">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
    </journal>
    <journal> ... </journal>
</library>
```

(Advanced Axes - Examples - ancestor)

```
<library>
    <journal> ... </journal>
    <journal>
        <article id="article.1">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.2">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.3">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
    </journal>
    <journal> ... </journal>
</library>
```

(Advanced Axes - Examples - following-sibling)

```
<library>
    <journal> ... </journal>
    <journal>
        <article id="article.1">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.2">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.3">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
    </journal>
    <journal> ... </journal>
</library>
```

(Advanced Axes - Examples - preceding-sibling)

```
<library>
    <journal> ... </journal>
    <journal>
        <article id="article.1">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.2">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.3">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
    </journal>
    <journal> ... </journal>
</library>
```

(Advanced Axes - Examples - following)

```
<library>
    <journal> ... </journal>
    <journal>
        <article id="article.1">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.2">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.3">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
    </journal>
    <journal> ... </journal>
</library>
```

(Advanced Axes - Examples - preceding)

```
<library>
    <journal> ... </journal>
    <journal>
        <article id="article.1">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.2">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.3">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
    </journal>
    <journal> ... </journal>
</library>
```

(Advanced Axes - Examples - attribute)

```
<library>
    <journal> ... </journal>
    <journal>
        <article id="article.1">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.2">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.3">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
    </journal>
    <journal> ... </journal>
</library>
```

(Advanced Axes - Examples - self)

```
<library>
    <journal> ... </journal>
    <journal>
        <article id="article.1">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.2">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.3">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
    </journal>
    <journal> ... </journal>
</library>
```

(Advanced Axes - Examples - descendant-or-self)

```
<library>
    <journal> ... </journal>
    <journal>
        <article id="article.1">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.2">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.3">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
    </journal>
    <journal> ... </journal>
</library>
```

(Advanced Axes - Examples - ancestor-or-self)

```
<library>
    <journal> ... </journal>
    <journal>
        <article id="article.1">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.2">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
        <article id="article.3">
            <author>
                <last></last>
                <first></first>
            </author>
        </article>
    </journal>
    <journal> ... </journal>
</library>
```

Node Types

Most of the previous examples have dealt with elements and attributes, but XPath defines several types of nodes that can be matched by expressions.

- **comment** Matches any comment node.

```
//comment()
```

- **text** Matches text nodes.

```
/journal/article/title/text()
```

- **processing-instruction** Matches processing-instructions.

```
//processing-instruction( 'template' )
```

- **node** Matches any node.

```
/journal/article/author/node()
```

(Node Types - Implicit)

Implicit node types addressable by names include:

- **element** The default node-type when performing name-based matches:

```
/journal/article/author
```

- **attribute** The nodes matched by steps along the attribute axis.

```
/journal/article/attribute::id
```

```
/journal/article/@id
```

- **namespace** The nodes matched by steps along the namespace axis.

```
/journal/namespace::*
```

Abbreviations

The XPath syntax includes a few abbreviations to make authoring expressions easier. Implicitly, the child axis is used for any name-based matches.

```
/journal/article
```

```
/child::journal/child::article
```

The '`@`' symbol is used as an abbreviation for the attribute axis.

```
/journal/article/@id
```

```
/journal/article/attribute::id
```

(Abbreviations)

The ‘.’ character is an abbreviation for self::node().

```
/journal/article[./@id='article.1']
```

The ‘//’ sequence is an abbreviation for descendant-or-self::node() which allows for matching nodes with arbitrary nodes in-between.

```
//author/last[.='Strachan']

/descendant-or-self::node()/author/last[.='Strachan']
```

The ‘*’ character matches any element or attribute name.

```
/journal/*[author/last='Strachan']

/journal/@*
```

Navigator

Adapting models for XPath

Navigation

Since XPath expressions steps are evaluated in relation to a context node across a particular axis of 'travel', the main aspects of navigating an object-model involves iterating over members of an axis.

The Navigator interface has a method for each axis in the form of:

```
Iterator get${AXIS}AxisIterator(Object contextNode)
    throws UnsupportedAxisException;
```

(Navigation - Axis Iterators)

```
package org.jaxen;

public interface Navigator
{
    Iterator getChildAxisIterator(Object contextNode)
        throws UnsupportedAxisException;

    Iterator getDescendantAxisIterator(Object contextNode)
        throws UnsupportedAxisException;

    Iterator getParentAxisIterator(Object contextNode)
        throws UnsupportedAxisException;

    Iterator getAncestorAxisIterator(Object contextNode)
        throws UnsupportedAxisException;

    Iterator getFollowingSiblingAxisIterator(Object contextNode)
        throws UnsupportedAxisException;

    Iterator getPrecedingSiblingAxisIterator(Object contextNode)
        throws UnsupportedAxisException;
```

continues. . .

(Navigator - Axis Iterator - continued)

. . . continued

```
    Iterator getFollowingAxisIterator(Object contextNode)
        throws UnsupportedAxisException;

    Iterator getPrecedingAxisIterator(Object contextNode)
        throws UnsupportedAxisException;

    Iterator getAttributeAxisIterator(Object contextNode)
        throws UnsupportedAxisException;

    Iterator getNamespaceAxisIterator(Object contextNode)
        throws UnsupportedAxisException;

    Iterator getSelfAxisIterator(Object contextNode)
        throws UnsupportedAxisException;

    Iterator getDescendantOrSelfAxisIterator(Object contextNode)
        throws UnsupportedAxisException;

    Iterator getAncestorOrSelfAxisIterator(Object contextNode)
        throws UnsupportedAxisException;
}
```

(Navigation - DefaultNavigator)

Since many of the axes are composite axes that can be synthesized from a sub-set, jaxen provides the DefaultNavigator which is a useful base class for model-specific navigators.

If a model-specific navigator implements getParentAxisIterator(...) then the DefaultNavigator can synthesize a useful default getAncestorAxisIterator(...).

(Navigation - DefaultNavigator - Axis Synthesis)

The following axes can be synthesized:

- **descendant** Built from child recursively.
- **ancestor** Built from parent recursively.
- **self** Completely synthetic.
- **descendant-or-self** Built from child recursively.
- **ancestor-or-self** Built from parent recursively.
- **following** Built from parent and child.
- **preceding** Built from parent and child.
- **following-sibling** Built from parent and child.
- **preceding-sibling** Built from parent and child.

Much of the axes are defined by purely providing implementation for accessing parent and child relationships.

(Navigation - Node types)

Each method is responsible for inspecting the contextNode parameter object and returning an Iterator over the axis in relation to the context-node object. If the object-model as a whole does not support a particular axis of travel, UnsupportedAxisException may be thrown.

The core jaxen engine will ensure that the methods are not called with a non-sensical context. For example, getAttributeAxisIterator(...) will never be called with a comment node as the parameter.

The Navigator provides methods to allow the core engine to determine the node's type.

(Navigation - Node types - Tests)

```
package org.jaxen;

public interface Navigator
{
    boolean isDocument(Object object);

    boolean isElement(Object object);

    boolean isAttribute(Object object);

    boolean isNamespace(Object object);

    boolean isComment(Object object);

    boolean isText(Object object);

    boolean isProcessingInstruction(Object object);
}
```

(Navigation - Inspection)

The core jaxen engine requires a way to inspect nodes for various properties, such as names, namespace URIs, and string values.

```
package org.jaxen;

public interface Navigator
{
    String getElementName(Object element);
    String getElementNamespaceUri(Object element);
    String getAttributeName(Object attr);
    String getAttributeNameNamespaceUri(Object attr);
    String getProcessingInstructionTarget(Object pi);
    String getProcessingInstructionData(Object pi);

    String getCommentStringValue(Object comment);
    String getElementStringValue(Object element);
    String getAttributeStringValue(Object attr);
    String getNamespaceStringValue(Object ns);
    String getTextStringValue(Object text);
    String getTextStringValue(Object text);
}
```

(Navigation - Example Navigator)

Here are some examples of implementations from the DocumentNavigator for dom4j.

(Navigation - Example Navigator - Axis Iterators)

```
public Iterator getChildAxisIterator(Object contextNode)
{
    if ( contextNode instanceof Branch )
    {
        Branch node = (Branch) contextNode;

        return node.nodeIterator();
    }

    return null;
}

public Iterator getAttributeAxisIterator(Object contextNode)
{
    if ( ! ( contextNode instanceof Element ) )

        return null;


    Element elem = (Element) contextNode;

    return elem.attributeIterator();
}
```

(Navigation - Example Navigator - Node Types)

```
public boolean isText(Object obj)
{
    return ( obj instanceof Text
             ||
             obj instanceof CDATA );
}

public boolean isAttribute(Object obj)
{
    return obj instanceof Attribute;
}

public boolean isProcessingInstruction(Object obj)
{
    return obj instanceof ProcessingInstruction;
}
```

(Navigation - Example Navigator - Inspection)

```
public String getAttributeName(Object obj)
{
    Attribute attr = (Attribute) obj;

    return attr.getName();
}

public String getAttributeNamespaceUri(Object obj)
{
    Attribute attr = (Attribute) obj;

    String uri = attr.getNamespaceURI();
    if ( uri != null && uri.length() == 0 )
        return null;
    else
        return uri;
}

public String getNamespaceStringValue(Object obj)
{
    Namespace ns = (Namespace) obj;

    return ns.getURI();
}
```

Thanks

I'd particularly wish to thank Pete Kazmier and Jeffrey Brekke for their diligent review of this presentation. Any errors that remain herein are purely my own responsibility.

Colophon

FoilTEX & LATEX 2$_\varepsilon$ were used in the production of these slides.

Images were produced via `dia`, exported to `EPS` and converted to `PDF` for inclusion using the `epstopdf` utility.

This slide deck is 100% Microsoft-free and produced using only open-source software.