

# Using the Java language NamespaceContext object with XPath

## Methods to resolve namespaces with the Java API

Skill Level: Intermediate

[Holger Kraus \(holger.kraus@de.ibm.com\)](mailto:holger.kraus@de.ibm.com)  
IT-Specialist  
IBM

19 May 2009

If you want to use namespaces in XPath expressions, you have to provide the link of the used prefix to the URI of the namespace. This article describes three variants of providing the prefix to namespace mapping. It contains example code to make it easy to code your own NamespaceContext.

## Prerequisites and the example

In this article, I assume that you are familiar with the technical details described in "Evaluating XPath from the Java™ platform" by Brett McLaughlin. If you are not sure how to run Java programs using XPath, please refer to the Brett's article (see [Resources](#) for a link to the article.) The same is true for the API needed to load an XML file and to evaluate an XPath expression.

You will use the following XML file for all examples:

### Listing 1. Example XML

```
<?xml version="1.0" encoding="UTF-8"?>

<books:booklist
  xmlns:books="http://univNaSpResolver/booklist"
  xmlns="http://univNaSpResolver/book"
  xmlns:fiction="http://univNaSpResolver/fictionbook">
  <science:book xmlns:science="http://univNaSpResolver/sciencebook">
```

```
<title>Learning XPath</title>
<author>Michael Schmidt</author>
</science:book>
<fiction:book>
  <title>Faust I</title>
  <author>Johann Wolfgang von Goethe</author>
</fiction:book>
<fiction:book>
  <title>Faust II</title>
  <author>Johann Wolfgang von Goethe</author>
</fiction:book>
</books:booklist>
```

This XML example has three namespaces declared in the root element and one declared on an element deeper in the structure. You will see what differences result from this setup.

### Frequently used acronyms

- API: application programming interface
- DOM: Document Object Model
- URI: Universal Resource Identifier
- XHTML: Extensible Hypertext Markup Language
- XML: Extensible Markup Language
- XSD: XML Schema Definition
- XSLT: Extensible Stylesheet Language Transformations

The second interesting thing about this XML example is that the element `booklist` has three children, all named `book`. But the first child has the namespace `science`, while the following children have the namespace `fiction`. This means that these elements are completely different to XPath. You will see the consequences in the examples below.

A word about the sample source code: the code is not optimized for maintenance, but for readability. This means that it has some redundancies. The output is produced in the simplest way through `System.out.println()`. All lines of code concerning output are abbreviated with `'...'` in the article. Also, I do not cover the helper methods in this article, but they are included in the download file (see [Download](#)).

## Theoretical background

What is the sense of namespaces and why care about them? A namespace is a part of the identifier for an element or attribute. You can have elements or attributes with the same local name, but different namespaces. They are completely different. See

the example above (`science:book` and `fiction:book`). You need namespaces to resolve naming conflicts if you combine XML files from different sources. Take, for example, an XSLT file. It consists of elements of the XSLT namespace, elements from your own namespace, and (often) elements of the XHTML namespace. Using namespaces you can avoid ambiguities concerning elements with the same local name.

The namespace is defined by the URI (in this example, `http://univNaSpResolver/booklist`). To avoid the use of this long string, you define a prefix that is associated with this URI (in the example, `books`). Please remember that the prefix is like a variable: its name does not matter. If two prefixes reference the same URI, the namespace of the prefixed elements will be the same (see Example 1 in [Listing 5](#) for an example of this).

An XPath expression uses prefixes (for example, `books:booklist/science:book`) and, you have to provide the URI associated with each prefix. This is where the `NamespaceContext` comes in. It does exactly that.

This article explains the different ways to provide the mapping between the prefix and the URI.

In the XML file, the mapping is provided by the `xmlns` attributes like:  
`xmlns:books="http://univNaSpResolver/booklist"` or  
`xmlns="http://univNaSpResolver/book"` (the default namespace).

## The necessity of providing namespace resolution

If you have XML that uses namespaces, an XPath expression will fail if you don't provide a `NamespaceContext`. Example 0 in [Listing 2](#) shows that case. The XPath object is constructed and evaluated on the loaded XML document. First, try to write the expression without any namespace prefixes (`result1`). In the second part, write the expression with namespace prefixes (`result2`).

### Listing 2. Example 0 without namespace resolution

```
private static void example0(Document example)
    throws XPathExpressionException, TransformerException {
    sysout("\n*** Zero example - no namespaces provided ***");

    XPath xPath = XPathFactory.newInstance().newXPath();

    ...

    NodeList result1 = (NodeList) xPath.evaluate("booklist/book", example,
        XPathConstants.NODESET);

    ...

    NodeList result2 = (NodeList) xPath.evaluate(
        "books:booklist/science:book", example, XPathConstants.NODESET);

    ...
}
```

```
}
```

This results in the following output.

### Listing 3. Output from example 0

```
*** Zero example - no namespaces provided ***
First try asking without namespace prefix:
--> booklist/book
Result is of length 0
Then try asking with namespace prefix:
--> books:booklist/science:book
Result is of length 0
The expression does not work in both cases.
```

In both cases, the XPath evaluation does not return any node and there is no exception. XPath cannot find a node because the mapping of the prefixes to the URIs is missing.

## Hardcoded namespace resolution

It is possible to supply the namespaces as hard-coded values which might look like the class in [Listing 4](#):

### Listing 4. Hardcoded namespace resolution

```
public class HardcodedNamespaceResolver implements NamespaceContext {

    /**
     * This method returns the uri for all prefixes needed. Wherever possible
     * it uses XMLConstants.
     *
     * @param prefix
     * @return uri
     */
    public String getNamespaceURI(String prefix) {
        if (prefix == null) {
            throw new IllegalArgumentException("No prefix provided!");
        } else if (prefix.equals(XMLConstants.DEFAULT_NS_PREFIX)) {
            return "http://univNaSpResolver/book";
        } else if (prefix.equals("books")) {
            return "http://univNaSpResolver/booklist";
        } else if (prefix.equals("fiction")) {
            return "http://univNaSpResolver/fictionbook";
        } else if (prefix.equals("technical")) {
            return "http://univNaSpResolver/sciencebook";
        } else {
            return XMLConstants.NULL_NS_URI;
        }
    }

    public String getPrefix(String namespaceURI) {
        // Not needed in this context.
        return null;
    }
}
```

```

    }

    public Iterator getPrefixes(String namespaceURI) {
        // Not needed in this context.
        return null;
    }
}

```

Please notice that the namespace `http://univNaSpResolver/sciencebook` is bound to the prefix `technical` (not `science` as before). You will see the consequences in the [example](#) below (Listing 6). In [Listing 5](#), the code using this resolver uses the new prefix.

### Listing 5. Example 1 with hardcoded namespace resolution

```

private static void example1(Document example)
    throws XPathExpressionException, TransformerException {
    sysout("\n*** First example - namespacelookup hardcoded ***");

    XPath xPath = XPathFactory.newInstance().newXPath();
    xPath.setNamespaceContext(new HardcodedNamespaceResolver());

    ...

    NodeList result1 = (NodeList) xPath.evaluate(
        "books:booklist/technical:book", example,
        XPathConstants.NODESET);

    ...

    NodeList result2 = (NodeList) xPath.evaluate(
        "books:booklist/fiction:book", example, XPathConstants.NODESET);

    ...

    String result = xPath.evaluate("books:booklist/technical:book/:author",
        example);

    ...
}

```

This is the output from this example.

### Listing 6. Output from example 1

```

*** First example - namespacelookup hardcoded ***
Using any namespaces results in a NodeList:
--> books:booklist/technical:book
Number of Nodes: 1
<?xml version="1.0" encoding="UTF-8"?>
  <science:book xmlns:science="http://univNaSpResolver/sciencebook">
    <title xmlns="http://univNaSpResolver/book">Learning XPath</title>
    <author xmlns="http://univNaSpResolver/book">Michael Schmidt</author>
  </science:book>
--> books:booklist/fiction:book
Number of Nodes: 2
<?xml version="1.0" encoding="UTF-8"?>
  <fiction:book xmlns:fiction="http://univNaSpResolver/fictionbook">
    <title xmlns="http://univNaSpResolver/book">Faust I</title>
    <author xmlns="http://univNaSpResolver/book">Johann Wolfgang von Goethe</author>
  </fiction:book>
<?xml version="1.0" encoding="UTF-8"?>
  <fiction:book xmlns:fiction="http://univNaSpResolver/fictionbook">

```

```

<title xmlns="http://univNaSpResolver/book">Faust II</title>
<author xmlns="http://univNaSpResolver/book">Johann Wolfgang von Goethe</author>
</fiction:book>
The default namespace works also:
--> books:booklist/technical:book/:author
Michael Schmidt

```

As you see, XPath finds the nodes now. The advantage is that you can rename the prefixes as you wish, which is what I did with the prefix `science`. The XML file contains the prefix `science`, while XPath uses another prefix, `technical`. Because the URIs are the same, the nodes are found by XPath. The disadvantage is that you have to maintain the namespaces in more places: the XML, perhaps the XSD, the XPath expression, and the namespace context.

## Read the namespaces from the document

The namespaces and their prefixes are documented in the XML file, so you can use them from there. The simplest way to do this is to delegate the lookup to the document.

### Listing 7. Namespace resolution directly from the document

```

public class UniversalNamespaceResolver implements NamespaceContext {
    // the delegate
    private Document sourceDocument;

    /**
     * This constructor stores the source document to search the namespaces in
     * it.
     *
     * @param document
     *        source document
     */
    public UniversalNamespaceResolver(Document document) {
        sourceDocument = document;
    }

    /**
     * The lookup for the namespace uris is delegated to the stored document.
     *
     * @param prefix
     *        to search for
     * @return uri
     */
    public String getNamespaceURI(String prefix) {
        if (prefix.equals(XMLConstants.DEFAULT_NS_PREFIX)) {
            return sourceDocument.lookupNamespaceURI(null);
        } else {
            return sourceDocument.lookupNamespaceURI(prefix);
        }
    }

    /**
     * This method is not needed in this context, but can be implemented in a
     * similar way.
     */
    public String getPrefix(String namespaceURI) {

```

```

        return sourceDocument.lookupPrefix(namespaceURI);
    }

    public Iterator getPrefixes(String namespaceURI) {
        // not implemented yet
        return null;
    }
}

```

Remember these things:

- If the document is changed before XPath is used, this change will still be reflected in the lookup of the namespace, because the delegation is done when needed using the current version of the document.
- The lookup for namespaces or prefixes is done in the ancestors of the used node, in our case the node `sourceDocument`. This means, with the code provided, you only get the namespaces declared on the root node. The namespace `science` in our example is not found.
- The lookup is called when XPath evaluates, so it consumes some extra time.

This is the example code:

### Listing 8. Example 2 with namespace resolution directly from the document

```

private static void example2(Document example)
    throws XPathExpressionException, TransformerException {
    sysout("\n*** Second example - namespacelookup delegated to document ***");

    XPath xPath = XPathFactory.newInstance().newXPath();
    xPath.setNamespaceContext(new UniversalNamespaceResolver(example));

    try {
    ...
        NodeList result1 = (NodeList) xPath.evaluate(
            "books:booklist/science:book", example,
            XPathConstants.NODESET);
    ...
    } catch (XPathExpressionException e) {
    ...
    }
    ...
    NodeList result2 = (NodeList) xPath.evaluate(
        "books:booklist/fiction:book", example, XPathConstants.NODESET);
    ...
    String result = xPath.evaluate(
        "books:booklist/fiction:book[1]/:author", example);
    ...
    }
}

```

The output from the example is:

## Listing 9. Output from example 2

```

*** Second example - namespacelookup delegated to document ***
Try to use the science prefix: no result
--> books:booklist/science:book
The resolver only knows namespaces of the first level!
To be precise: Only namespaces above the node, passed in the constructor.
The fiction namespace is such a namespace:
--> books:booklist/fiction:book
Number of Nodes: 2
<?xml version="1.0" encoding="UTF-8"?>
  <fiction:book xmlns:fiction="http://univNaSpResolver/fictionbook">
    <title xmlns="http://univNaSpResolver/book">Faust I</title>
    <author xmlns="http://univNaSpResolver/book">Johann Wolfgang von Goethe</author>
  </fiction:book>
<?xml version="1.0" encoding="UTF-8"?>
  <fiction:book xmlns:fiction="http://univNaSpResolver/fictionbook">
    <title xmlns="http://univNaSpResolver/book">Faust II</title>
    <author xmlns="http://univNaSpResolver/book">Johann Wolfgang von Goethe</author>
  </fiction:book>
The default namespace works also:
--> books:booklist/fiction:book[1]/:author
Johann Wolfgang von Goethe

```

As you see in the output, the namespace declared on the `book` element with the prefix `science` is not resolved. The `evaluate` method throws an `XPathExpressionException`. To get around this problem, you might extract the node `science:book` from the document and use this node as the delegate. But this means extra parsing of the document and is not elegant.

## Read the namespaces from the document and cache them

This next version of the `NamespaceContext` is better. It reads the namespaces only one time in advance in the constructor. Every call for a namespace is answered from a cache. As a consequence, a change in the document does not matter since the list of namespaces is cached at Java object creation time.

## Listing 10. Caching the namespace resolution from the document

```

public class UniversalNamespaceCache implements NamespaceContext {
    private static final String DEFAULT_NS = "DEFAULT";
    private Map<String, String> prefix2Uri = new HashMap<String, String>();
    private Map<String, String> uri2Prefix = new HashMap<String, String>();

    /**
     * This constructor parses the document and stores all namespaces it can
     * find. If toplevelOnly is true, only namespaces in the root are used.
     *
     * @param document
     *        source document
     * @param topLevelOnly
     *        restriction of the search to enhance performance
     */
    public UniversalNamespaceCache(Document document, boolean topLevelOnly) {
        examineNode(document.getFirstChild(), topLevelOnly);
    }

```

```

        System.out.println("The list of the cached namespaces:");
        for (String key : prefix2Uri.keySet()) {
            System.out
                .println("prefix " + key + ": uri " + prefix2Uri.get(key));
        }
    }

/**
 * A single node is read, the namespace attributes are extracted and stored.
 *
 * @param node
 *         to examine
 * @param attributesOnly,
 *         if true no recursion happens
 */
private void examineNode(Node node, boolean attributesOnly) {
    NamedNodeMap attributes = node.getAttributes();
    for (int i = 0; i < attributes.getLength(); i++) {
        Node attribute = attributes.item(i);
        storeAttribute((Attr) attribute);
    }

    if (!attributesOnly) {
        NodeList children = node.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
            Node child = children.item(i);
            if (child.getNodeType() == Node.ELEMENT_NODE)
                examineNode(child, false);
        }
    }
}

/**
 * This method looks at an attribute and stores it, if it is a namespace
 * attribute.
 *
 * @param attribute
 *         to examine
 */
private void storeAttribute(Attr attribute) {
    // examine the attributes in namespace xmlns
    if (attribute.getNamespaceURI() != null
        && attribute.getNamespaceURI().equals(
            XMLConstants.XMLNS_ATTRIBUTE_NS_URI)) {
        // Default namespace xmlns="uri goes here"
        if (attribute.getNodeName().equals(XMLConstants.XMLNS_ATTRIBUTE)) {
            putInCache(DEFAULT_NS, attribute.getNodeValue());
        } else {
            // The defined prefixes are stored here
            putInCache(attribute.getLocalName(), attribute.getNodeValue());
        }
    }
}

private void putInCache(String prefix, String uri) {
    prefix2Uri.put(prefix, uri);
    uri2Prefix.put(uri, prefix);
}

/**
 * This method is called by XPath. It returns the default namespace, if the
 * prefix is null or "".
 *
 * @param prefix
 *         to search for
 * @return uri
 */
public String getNamespaceURI(String prefix) {

```

```

        if (prefix == null || prefix.equals(XMLConstants.DEFAULT_NS_PREFIX)) {
            return prefix2Uri.get(DEFAULT_NS);
        } else {
            return prefix2Uri.get(prefix);
        }
    }

    /**
     * This method is not needed in this context, but can be implemented in a
     * similar way.
     */
    public String getPrefix(String namespaceURI) {
        return uri2Prefix.get(namespaceURI);
    }

    public Iterator getPrefixes(String namespaceURI) {
        // Not implemented
        return null;
    }
}

```

Please note that there is debug output in the code. The attributes of each node are examined and stored. The children are not examined, because the boolean `toplevelOnly` in the constructor is set to `true`. If the boolean is set to `false`, the examination of the children will start after the attributes are stored. One thing to consider about the code: In DOM, the first node represents the document as a whole, so, to get the element `book` to read the namespaces, you have to go to the children exactly one time.

In this case, using `NamespaceContext` is rather simple:

### Listing 11. Example 3 with cached namespace resolution (toplevel only)

```

private static void example3(Document example)
    throws XPathExpressionException, TransformerException {
    System.out.println("Third example - namespaces of topLevel node cached");

    XPath xPath = XPathFactory.newInstance().newXPath();
    xPath.setNamespaceContext(new UniversalNamespaceCache(example, true));

    try {
        ...
        NodeList result1 = (NodeList) xPath.evaluate(
            "books:booklist/science:book", example,
            XPathConstants.NODESET);
        ...
    } catch (XPathExpressionException e) {
        ...
    }

    ...
    NodeList result2 = (NodeList) xPath.evaluate(
        "books:booklist/fiction:book", example, XPathConstants.NODESET);
    ...
    String result = xPath.evaluate(
        "books:booklist/fiction:book[1]/:author", example);
    ...
}

```

This results in the following output:

### Listing 12. Output from Example 3

```

*** Third example - namespaces of toplevel node cached ***
The list of the cached namespaces:
prefix DEFAULT: uri http://univNaSpResolver/book
prefix fiction: uri http://univNaSpResolver/fictionbook
prefix books: uri http://univNaSpResolver/booklist
Try to use the science prefix:
--> books:booklist/science:book
The cache only knows namespaces of the first level!
The fiction namespace is such a namespace:
--> books:booklist/fiction:book
Number of Nodes: 2
<?xml version="1.0" encoding="UTF-8"?>
  <fiction:book xmlns:fiction="http://univNaSpResolver/fictionbook">
    <title xmlns="http://univNaSpResolver/book">Faust I</title>
    <author xmlns="http://univNaSpResolver/book">Johann Wolfgang von Goethe</author>
  </fiction:book>
  <?xml version="1.0" encoding="UTF-8"?>
    <fiction:book xmlns:fiction="http://univNaSpResolver/fictionbook">
      <title xmlns="http://univNaSpResolver/book">Faust II</title>
      <author xmlns="http://univNaSpResolver/book">Johann Wolfgang von Goethe</author>
    </fiction:book>
The default namespace works also:
--> books:booklist/fiction:book[1]/:author
Johann Wolfgang von Goethe

```

This code only finds the namespaces of the root element. To be precise: the namespaces of the node passed into the method `examineNode` by the constructor. This speeds up the constructor because it does not have to iterate through the whole document. However, as you can see from the output, the `science` prefix cannot be resolved. The XPath expression leads to an exception (`XPathExpressionException`).

## Read the namespaces from the document and all its elements and cache them

This version reads all namespace declarations from the XML file. Now, even the XPath on the prefix `science` works. One situation makes this version complicated: If a prefix is overloaded (declared in nested elements on different URIs), the last one found wins. In the real world, this typically is not a problem.

Using `NamespaceContext` in this example is the same as in the previous example. The boolean `toplevelOnly` in the constructor has to be set to `false`.

### Listing 13. Example 4 with cached namespace resolution (all levels)

```

private static void example4(Document example)
    throws XPathExpressionException, TransformerException {

```

```

    sysout("\n*** Fourth example - namespaces all levels cached ***");

    XPath xPath = XPathFactory.newInstance().newXPath();
    xPath.setNamespaceContext(new UniversalNamespaceCache(example, false));
    ...
    NodeList result1 = (NodeList) xPath.evaluate(
        "books:booklist/science:book", example, XPathConstants.NODESET);
    ...
    NodeList result2 = (NodeList) xPath.evaluate(
        "books:booklist/fiction:book", example, XPathConstants.NODESET);
    ...
    String result = xPath.evaluate(
        "books:booklist/fiction:book[1]/:author", example);
    ...
}

```

This results in the following output:

#### Listing 14. Output from example 4

```

*** Fourth example - namespaces all levels cached ***
The list of the cached namespaces:
prefix science: uri http://univNaSpResolver/sciencebook
prefix DEFAULT: uri http://univNaSpResolver/book
prefix fiction: uri http://univNaSpResolver/fictionbook
prefix books: uri http://univNaSpResolver/booklist
Now the use of the science prefix works as well:
--> books:booklist/science:book
Number of Nodes: 1
<?xml version="1.0" encoding="UTF-8"?>
  <science:book xmlns:science="http://univNaSpResolver/sciencebook">
    <title xmlns="http://univNaSpResolver/book">Learning XPath</title>
    <author xmlns="http://univNaSpResolver/book">Michael Schmidt</author>
  </science:book>
The fiction namespace is resolved:
--> books:booklist/fiction:book
Number of Nodes: 2
<?xml version="1.0" encoding="UTF-8"?>
  <fiction:book xmlns:fiction="http://univNaSpResolver/fictionbook">
    <title xmlns="http://univNaSpResolver/book">Faust I</title>
    <author xmlns="http://univNaSpResolver/book">Johann Wolfgang von Goethe</author>
  </fiction:book>
<?xml version="1.0" encoding="UTF-8"?>
  <fiction:book xmlns:fiction="http://univNaSpResolver/fictionbook">
    <title xmlns="http://univNaSpResolver/book">Faust II</title>
    <author xmlns="http://univNaSpResolver/book">Johann Wolfgang von Goethe</author>
  </fiction:book>
The default namespace works also:
--> books:booklist/fiction:book[1]/:author
Johann Wolfgang von Goethe

```

## Conclusion

You can choose from several ideas for the implementation of the namespace resolution that might be better than hardcoding it:

- If your example is small and all namespaces are located in the top

element, delegating to the document will do.

- If you have bigger XML files with deep nesting and multiple XPath evaluations, it might be better to cache the list of namespaces.
- But if you don't have control over the XML file, and someone can send you any prefixes they wish, it might be better to be independent of their choices. You can code your own namespace resolution as in Example 1 (HardcodedNamespaceResolver), and use them in your XPath expressions.

In all other cases, the NamespaceContext resolved from the XML file can make your code more general and smaller.

## Downloads

Description	Name	Size	Download method
Source code for the examples	sourceCode.zip	7KB	<a href="#">HTTP</a>

[Information about download methods](#)

# Resources

## Learn

- [Evaluating XPath from the Java platform](#) (Brett McLaughlin, developerWorks, July 2008): Read this guide to using XPath with the Java API.
- [IBM XML certification](#): Find out how you can become an IBM-Certified Developer in XML and related technologies.
- [XML technical library](#): See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- [developerWorks technical events and webcasts](#): Stay current with technology in these sessions.
- [developerWorks podcasts](#): Listen to interesting interviews and discussions for software developers.

## Get products and technologies

- [IBM product evaluation versions](#): Download or [explore the online trials in the IBM SOA Sandbox](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

## Discuss

- [XML zone discussion forums](#): Participate in any of several XML-related discussions.
- [developerWorks XML zone: Share your thoughts](#): After you read this article, post your comments and thoughts in this forum. The XML zone editors moderate the forum and welcome your input.
- [developerWorks blogs](#): Check out these blogs and get involved in the [developerWorks community](#).

## About the author

Holger Kraus

Holger studied mathematics at the University of Bonn in Germany. Since 1996, he has worked at IBM Global Business Services and develops solutions for customers on the customer's site. His experience in XPath and Java programming results from the use of these techniques in his projects.