# XML & XPath essentials

by Magdalena Turska / @magdaturska

eXist Solutions 2016

Press **SPACE** to navigate through slides.

## XML

Extensible Markup Language is a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable.

Defined by the W3C's XML 1.0 Specification and by several other related specifications, all of which are free open standards.

Strictly speaking, XML is a metalanguage, that is, a language used to describe other languages (or, as we often say: vocabularies). TEI is one of the existing vocabularies expressed in XML.

## UNICODE

XML document is a **string of characters**.

Almost every legal Unicode character may appear in an XML document.

## PROCESSOR AND APPLICATION

The processor analyzes the markup and passes structured information to an application. The specification places requirements on what an XML processor must do and not do, but the application is outside its scope. The processor (as the specification calls it) is often referred to colloquially as an XML parser.

## MARKUP AND CONTENT

The characters making up an XML document are divided into **markup** and **content**, which may be distinguished by the application of simple syntactic rules.

- markup either begins with < and ends with a >
- or begins with the character & and ends with a ;
- strings of characters that are not markup are content

However, in a CDATA section, the delimiters <![CDATA[ and ] ]> are classified as markup, while the text between them is classified as content. In addition, whitespace before and after the outermost element is classified as markup.

## TAG

A markup construct that begins with < and ends with >.

Tags come in three flavors:

- **start-tags**; for example: <div>
- **end-tags**; for example: </div>
- **empty-element tags**; for example: <lb/>

## ELEMENT

A logical document component which either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag.

The characters between the start- and end-tags, if any, are the element's content, and may contain markup, including other elements, which are called **child elements**. An example of an element is
<salute>Hello, world.</salute>.
Another is <lb/>.

## ATTRIBUTE

A markup construct consisting of a name/value pair that exists within a start-tag or empty-element tag. In the example the element `img` has two attributes, `src` and `alt`:

```
<img src="madonna.jpg" alt='Foligno Madonna, by Raphael'/>
```

Another example would be

```
<step number="3">Connect A to B.</step>
```

where the name of the attribute is `number` and the value is `3`.

## ATTRIBUTES

An XML attribute can only have a single value and each attribute can appear at most once on each element.

Where a list of multiple values is desired, this must be done by encoding the list into a well-formed XML attribute with some format beyond what XML defines itself. Usually this is either a **comma or semi-colon delimited list** or, if the individual values are known not to contain spaces, a **space-delimited list** can be used.

```
<div class="inner greeting-box">Hello!</div>
```

where the attribute `class` has both the value `inner greeting-box` and also indicates the two CSS class names `inner` and `greeting-box`.

XML declaration XML documents may begin by declaring some information about themselves, as in the following example:

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

## WELL-FORMEDNESS

XML document must be a well-formed text – it needs to satisfy a list of syntax rules provided in the specification.

- The document contains only properly encoded legal **Unicode** characters
- None of the special syntax characters such as **< and &** appear except when performing their markup-delineation roles
- The begin, end, and empty-element tags that delimit the elements are **correctly nested**, with none missing and none overlapping
- The element tags are **case-sensitive**; the beginning and end tags must match exactly.
- A single **root** element contains all the other elements.
- Tag names cannot contain any of the characters !"#$%&'()*+,/;<=>?@[\]^`{|}~, nor a space character, and cannot start with -, ., or a numeric digit.

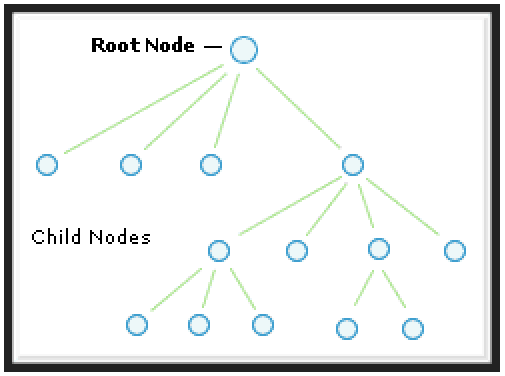
## XDM

The language is based on the XQuery and XPath Data Model (XDM) which uses a tree-structured model of the information content of an XML document, containing seven kinds of nodes:

**document nodes, elements, attributes, text nodes, comments, processing instructions, and namespaces.**

## XML NODES:

- The entire document is a document node
- Every XML element is an element node
- The text in the XML elements are text nodes
- Every attribute is an attribute node
- Comments are comment nodes
- Text is Always Stored in Text Nodes



A common error is to expect an element node to contain text. However, the text of an element node is

A common error is to expect an element node to contain text. However, the text of an element node is stored in a text node. In this example: <year>2005</year>, the element node <year></year>, holds a text node with the value "2005". "2005" is not the value of the <year></year> element!

## NODE PARENTS, CHILDREN, AND SIBLINGS

The nodes in the node tree have a hierarchical relationship to each other. The terms parent, child, and sibling are used to describe the relationships.

- Parent nodes have children.
- Children on the same level are called siblings
- In a node tree, the top node is called the root
- Every node, except the root, has exactly one parent node
- A node can have any number of children
- A leaf is a node with no children
- Siblings are nodes with the same parent

## NAMESPACES

- Namespace name is an identifier given to an XML vocabulary
- It looks a lot like URI, eg. TEI namespace name is **http://www.tei-c.org/ns/1.0**
- It doesn't mean that this URI actually **points** to something (like a schema), **this is just a name!**

*XML namespaces are used for providing uniquely named elements and attributes in an XML document. An XML instance may contain element or attribute names from more than one XML vocabulary. If each vocabulary is given a namespace, the ambiguity between identically named elements or attributes can be resolved.*

## MENTAL SHORTCUT TO NAMESPACES

If you assume that each XML vocabulary is basically a **language**, then namespace is a **name of this language**.

So saying that a particular element comes from a given namespace resolves the ambiguity, just like with natural languages, eg.

polish:fart means "luck" and is not smelly at all, contrary to english:fart

spanish:hola meaning "hi!" is not the same as polish:hola meaning "hey, man, now wait a minute!"

Back to XML html:div is not quite the same as tei:div, but they could be used in the same document

## HOW TO DECLARE A NAMESPACE FOR A DOCUMENT?

use xmlns attribute on root element, eg:

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
```

All descendants inherit the namespace from their parent, so no need to repeat it on all other elements

# HOW TO MIX ELEMENTS FROM DIFFERENT NAMESPACES?

use xmlns attribute on elements from 'embedded' namespaces, eg:

```
<include xmlns="http://www.w3.org/2001/XInclude"
href="elementsummary.xml"/>
```

declare namespace up top, give it a prefix and use prefixed element names, eg:

```
<TEI xmlns="http://www.tei-c.org/ns/1.0"
xmlns:skos="http://www.w3.org/2004/02/skos/core#"
xml:lang="en">
.... somewhere down in the document ....
<skos:symbol>blah</skos:symbol>
```

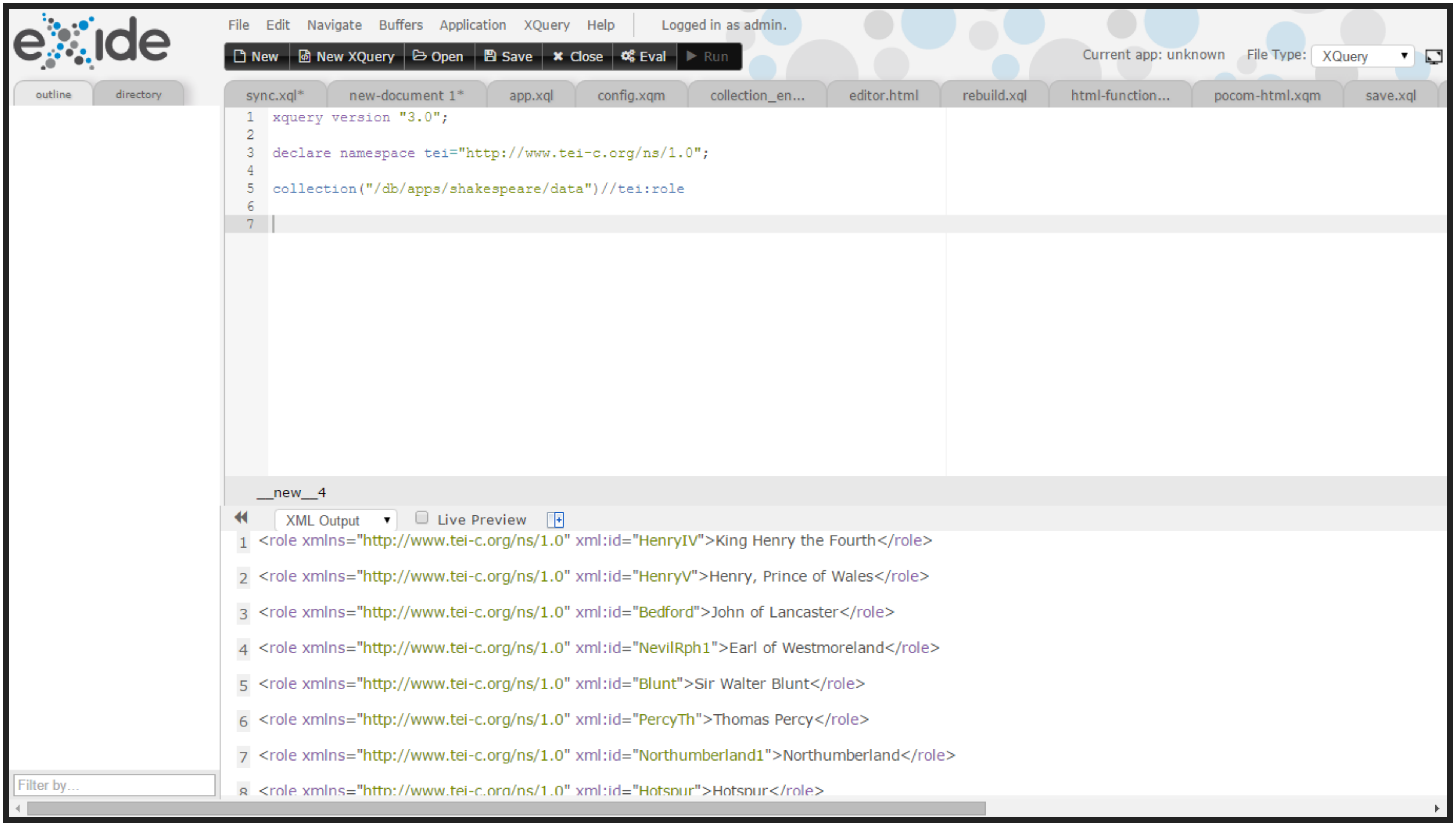
If you don't put xmlns attribute anywhere, all elements belong to the **default, empty namespace!**

```
<?xml version="1.0" encoding="UTF-8"?>
        <tei:bibl xmlns:tei="http://www.tei-c.org/ns/1.0">
                <title>eXist</title>
                <x:author xmlns:x="http://www.tei-c.org/ns/1.0">Adam
                <author>Erik Siegel</author>
                <publisher>O'Reilly</publisher>
                <date>2014</date>
                <idno type="ISBN">9781449337100</idno>
        </tei:bibl>
```

All elements above are in the TEI namespace

Different prefixes do not matter! tei:author and x:author are the same

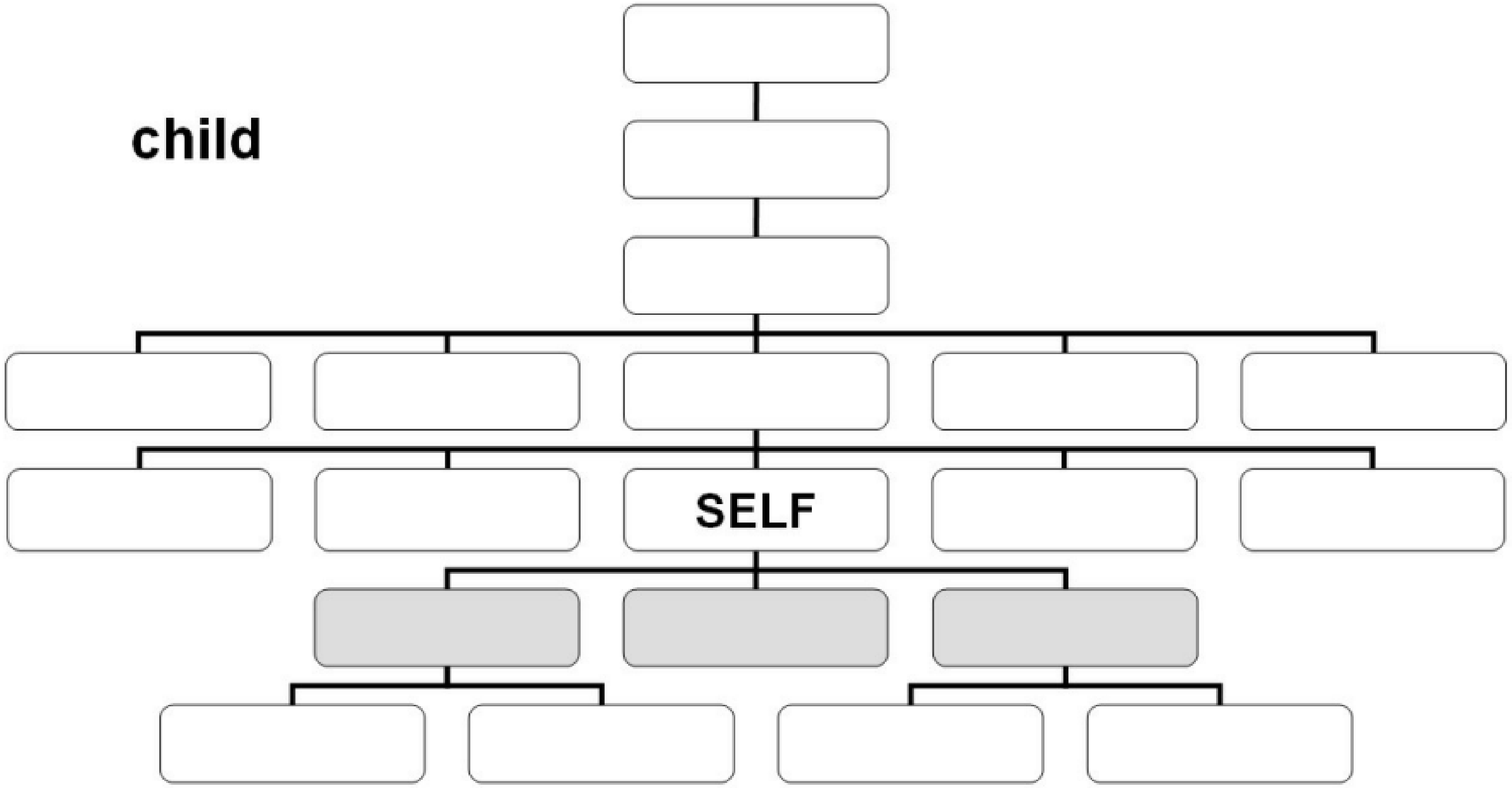
# XPATH



```
xquery version "3.0";

declare namespace tei="http://www.tei-c.org/ns/1.0";

collection("/db/apps/shakespeare/data")//tei:role

```

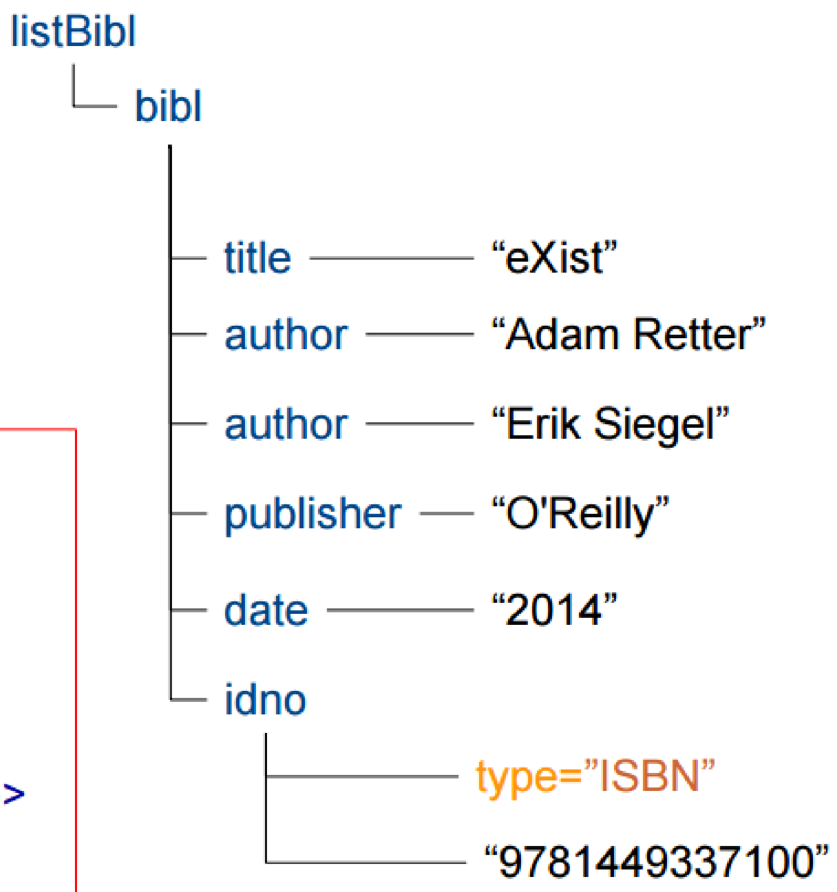
```
<role xmlns="http://www.tei-c.org/ns/1.0" xml:id="HenryIV">King Henry the Fourth</role>
<role xmlns="http://www.tei-c.org/ns/1.0" xml:id="HenryV">Henry, Prince of Wales</role>
<role xmlns="http://www.tei-c.org/ns/1.0" xml:id="Bedford">John of Lancaster</role>
<role xmlns="http://www.tei-c.org/ns/1.0" xml:id="NevilRph1">Earl of Westmoreland</role>
<role xmlns="http://www.tei-c.org/ns/1.0" xml:id="Blunt">Sir Walter Blunt</role>
<role xmlns="http://www.tei-c.org/ns/1.0" xml:id="PercyTh">Thomas Percy</role>
<role xmlns="http://www.tei-c.org/ns/1.0" xml:id="Northumberland1">Northumberland</role>
<role xmlns="http://www.tei-c.org/ns/1.0" xml:id="Hotspur">Hotspur</role>
```

## child



Find listBibl root element:

```
/listBibl
```



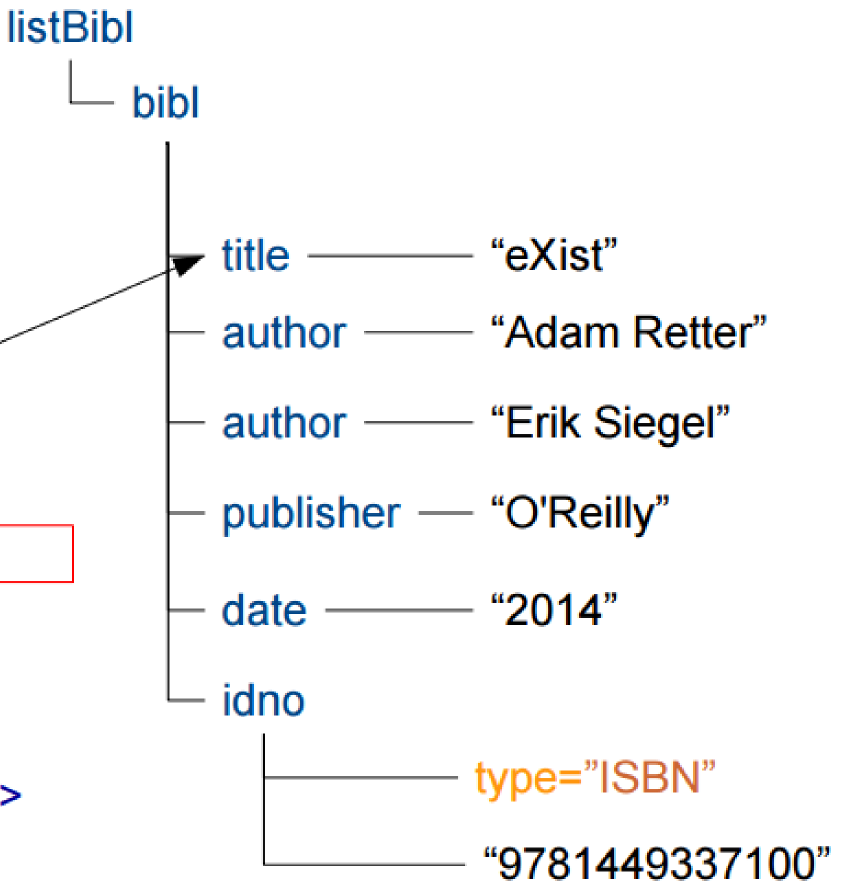
```
<?xml version="1.0" encoding="UTF-8"?>
<listBibl>
   <bibl>
      <title>eXist</title>
      <author>Adam Retter</author>
      <author>Erik Siegel</author>
      <publisher>O'Reilly</publisher>
      <date>2014</date>
      <idno type="ISBN">9781449337100</idno>
   </bibl>
</listBibl>
```

existsolutions

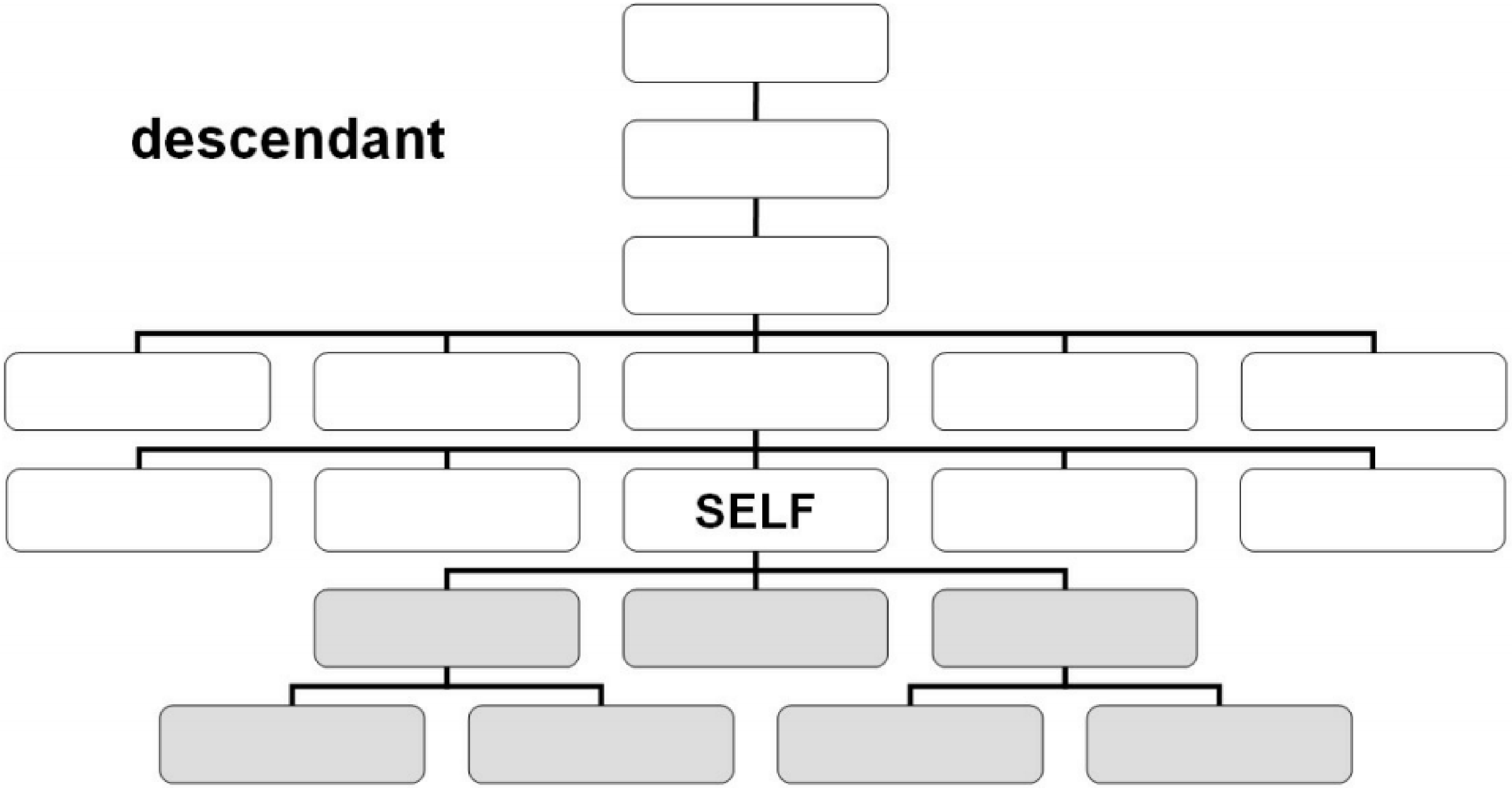
Find title child of bibl:

```
/listBibl/bibl/title
```

```
<?xml version="1.0" encoding="UTF-8"?>
<listBibl>
   <bibl>
      <title>eXist</title>
      <author>Adam Retter</author>
      <author>Erik Siegel</author>
      <publisher>O'Reilly</publisher>
      <date>2014</date>
      <idno type="ISBN">9781449337100</idno>
   </bibl>
</listBibl>
```



## descendant

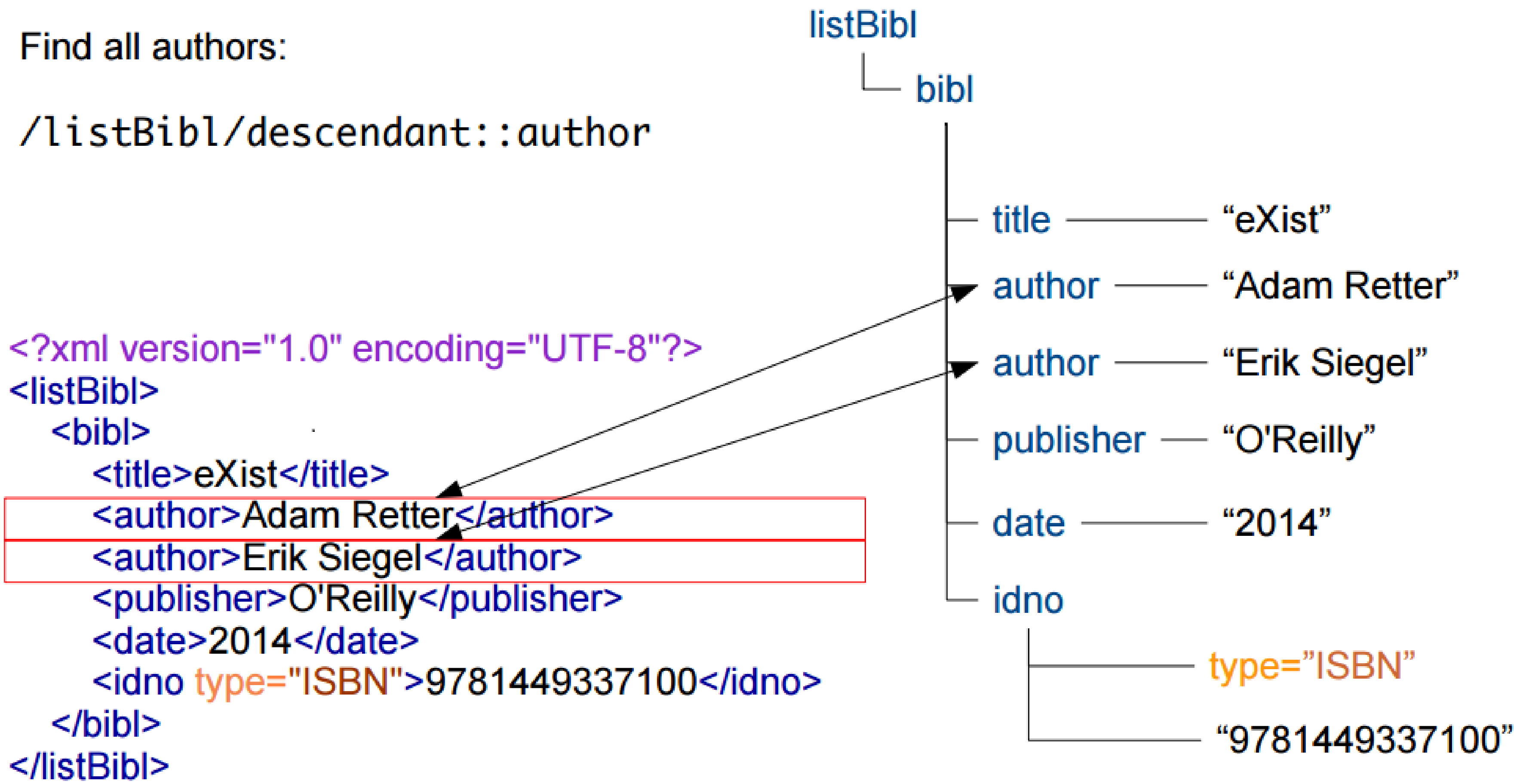


existsolutions

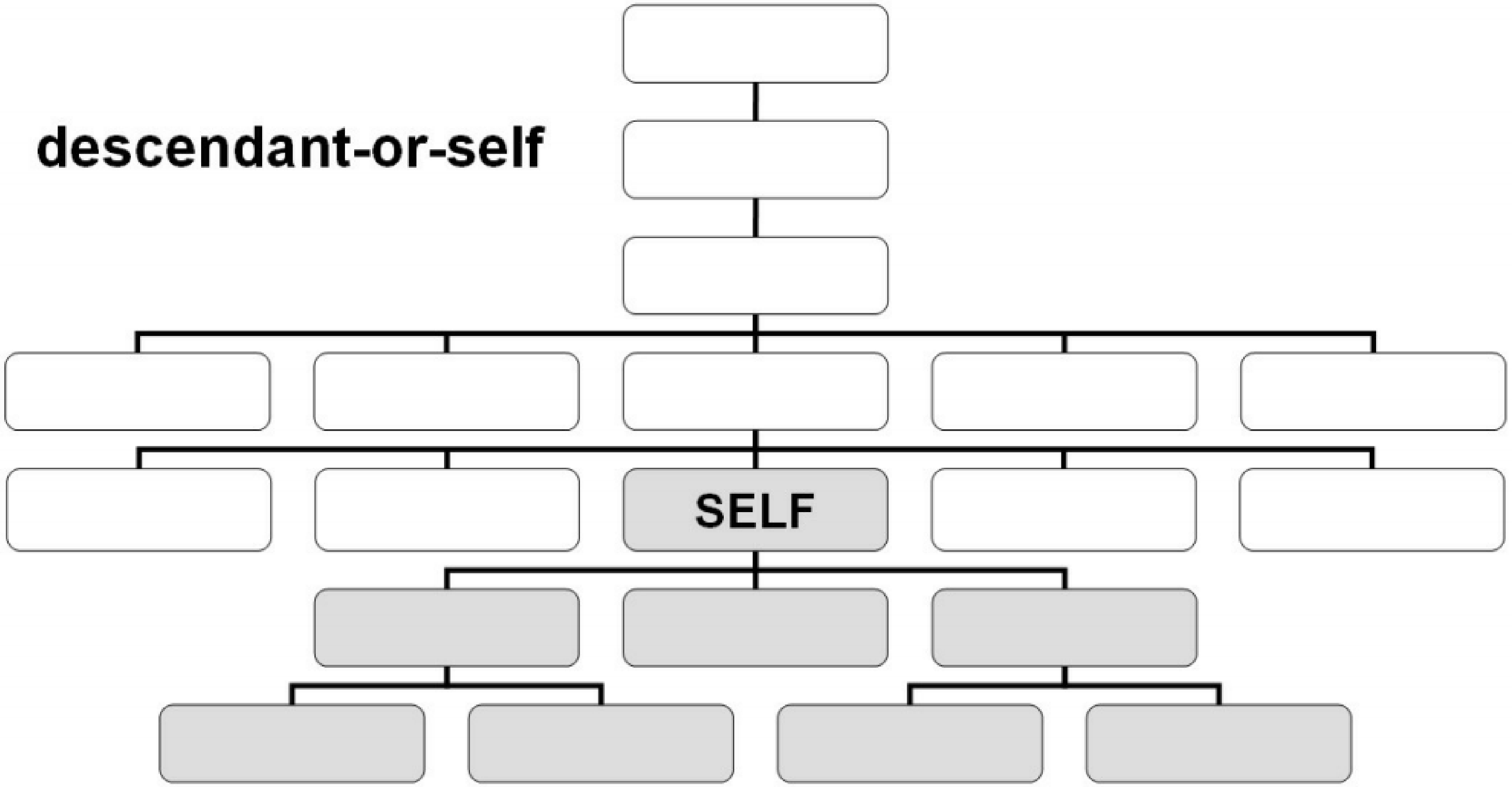
Find all authors:

```
/listBibl/descendant::author
```

```
<?xml version="1.0" encoding="UTF-8"?>
<listBibl>
   <bibl>
      <title>eXist</title>
      <author>Adam Retter</author>
      <author>Erik Siegel</author>
      <publisher>O'Reilly</publisher>
      <date>2014</date>
      <idno type="ISBN">9781449337100</idno>
   </bibl>
</listBibl>
```



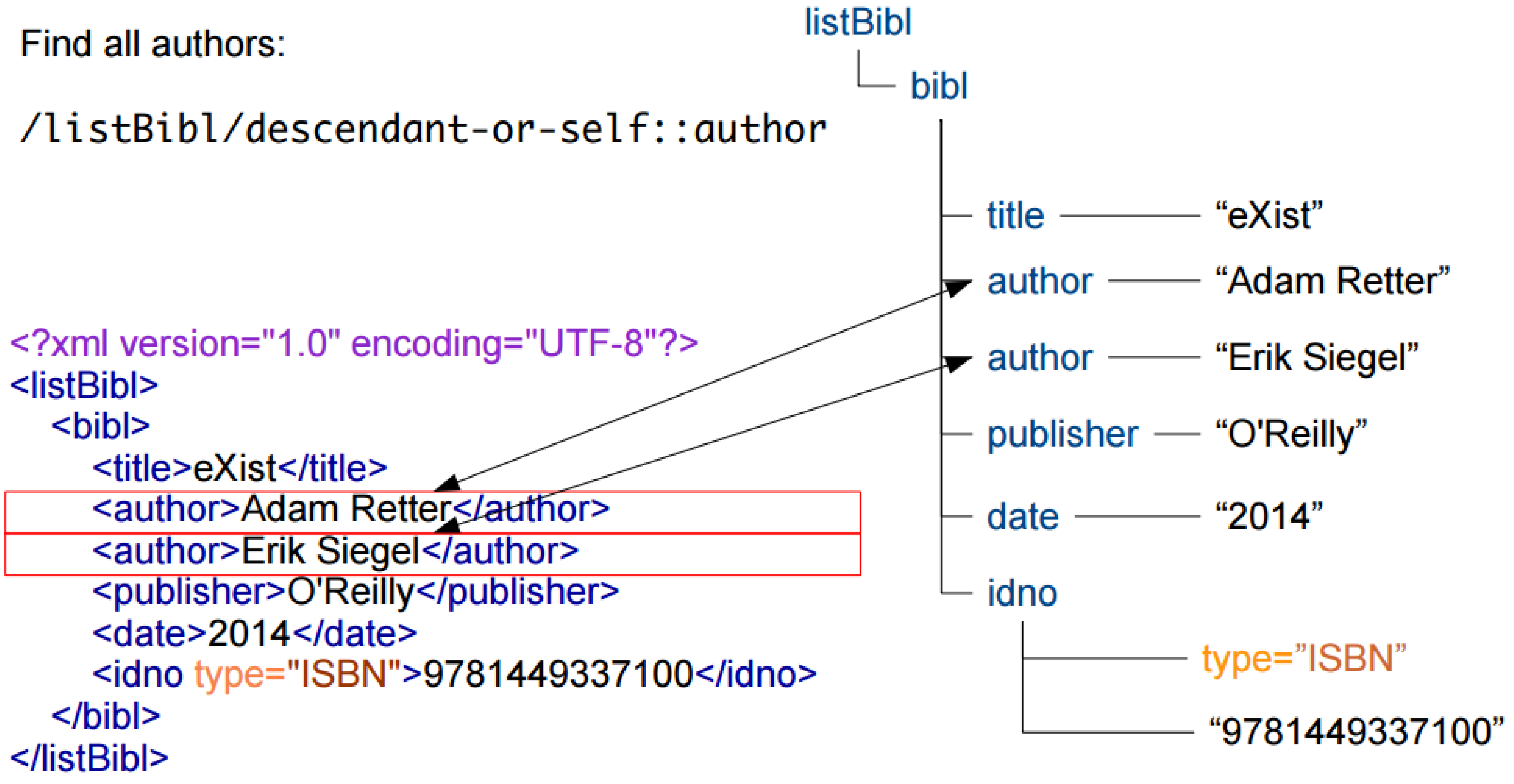
## descendant-or-self



existsolutions

Find all authors:

```
/listBibl/descendant-or-self::author
```



```xml
<?xml version="1.0" encoding="UTF-8"?>
<listBibl>
    <bibl>
        <title>eXist</title>
        <author>Adam Retter</author>
        <author>Erik Siegel</author>
        <publisher>O'Reilly</publisher>
        <date>2014</date>
        <idno type="ISBN">9781449337100</idno>
    </bibl>
</listBibl>
```

What does the following expression return?

```
/listBibl/bibl/descendant-or-self::bibl
```

## PREDICATES

- positional

```
/listBibl/bibl[1]
```

*Predicate expression evaluates to a positive number*

- filters

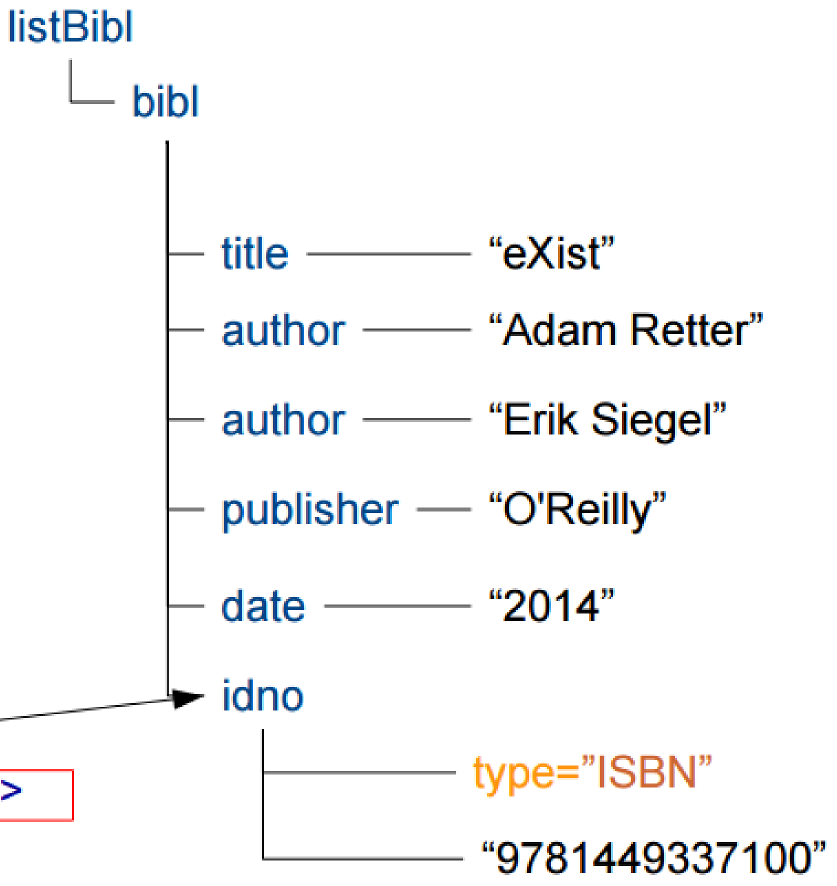
```
/descendant::bibl[title = "eXist"]
/descendant::bibl[title = "eXist"][date =
"2014"]
/descendant::bibl[title[. = "eXist"]]
/descendant::bibl[title[. = "eXist"]]/date
```

*Predicate expression interpreted as a boolean (true/false)*

Find idno matching type="ISBN":

```
/descendant::idno[@type = "ISBN"]
```

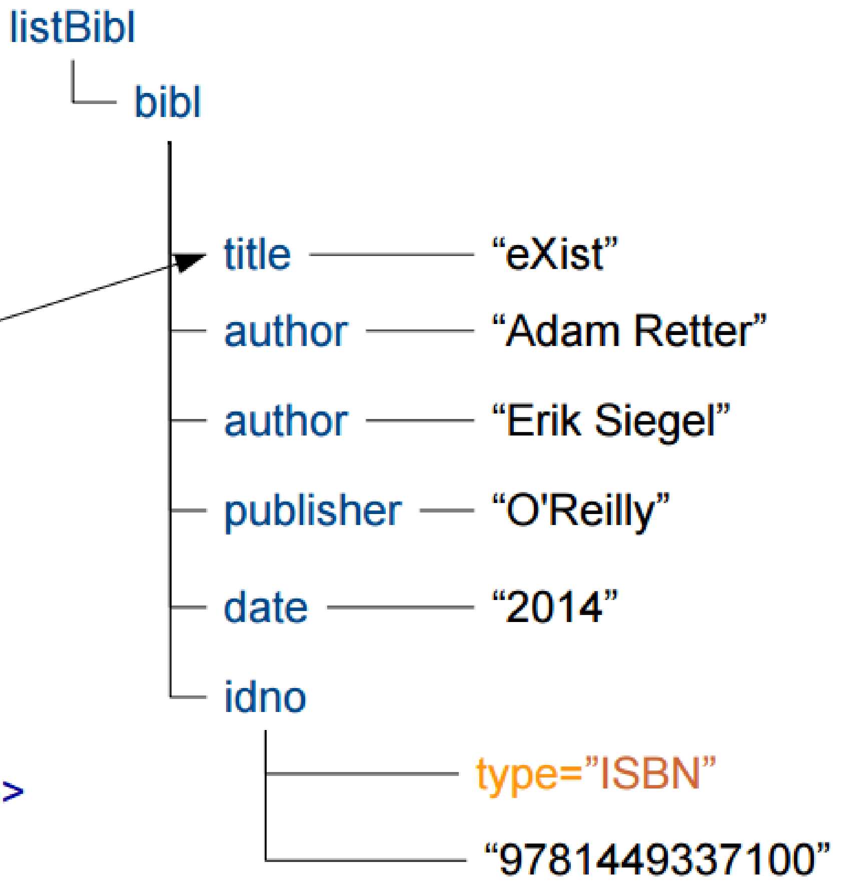
```
<?xml version="1.0" encoding="UTF-8"?>
<listBibl>
   <bibl>
      <title>eXist</title>
      <author>Adam Retter</author>
      <author>Erik Siegel</author>
      <publisher>O'Reilly</publisher>
      <date>2014</date>
      <idno type="ISBN">9781449337100</idno>
   </bibl>
</listBibl>
```



Get titles as text:

```
/descendant::title/text()
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<listBibl>
    <bibl>
        <title>eXist</title>
        <author>Adam Retter</author>
        <author>Erik Siegel</author>
        <publisher>O'Reilly</publisher>
        <date>2014</date>
        <idno type="ISBN">9781449337100</idno>
    </bibl>
</listBibl>
```



## NODE TESTS

- node() Selects any node (except attributes)
- attribute() Selects any attribute (shortcut: @*)
- element() Any element (shortcut: *)
- text() Selects text nodes

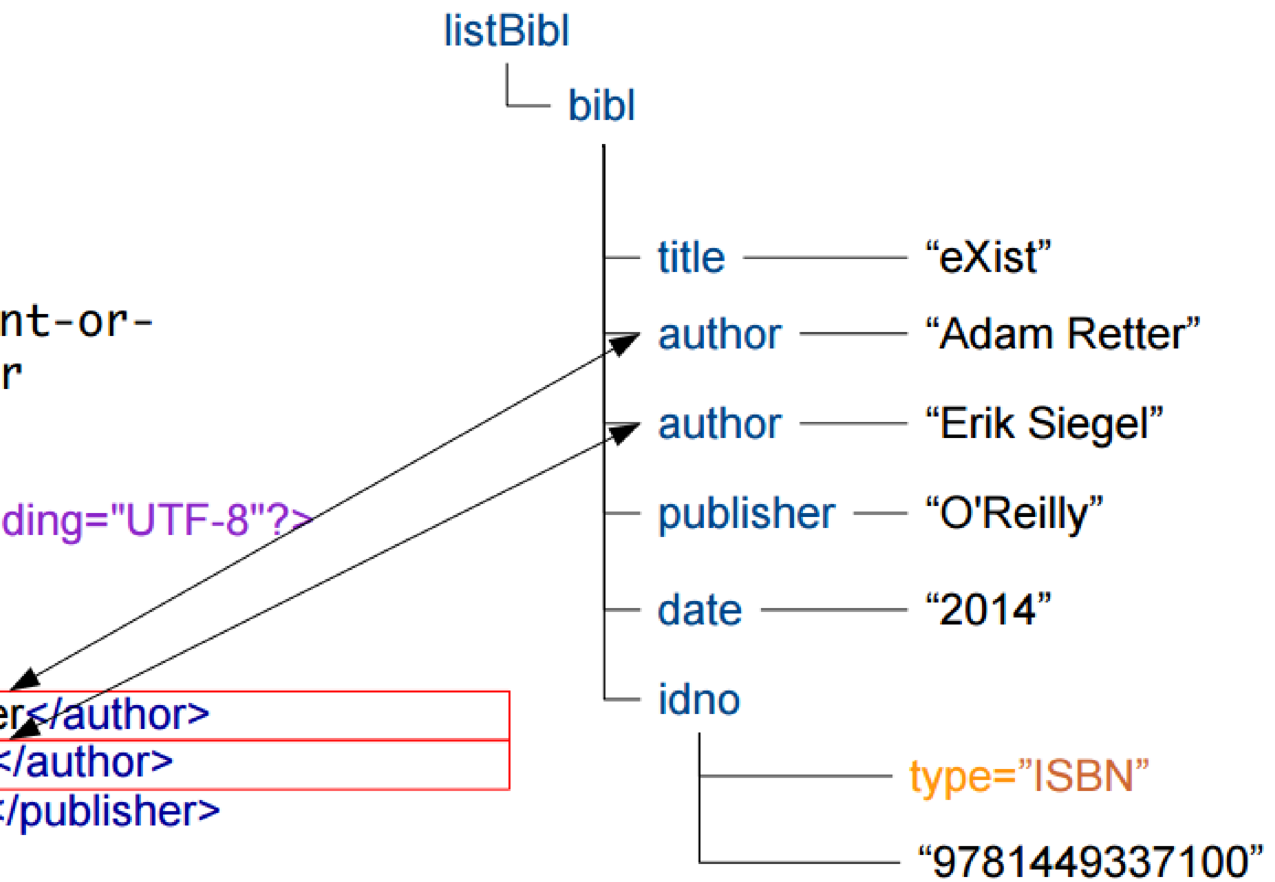
Find all authors:

```
/listBibl//author
```

Same as:

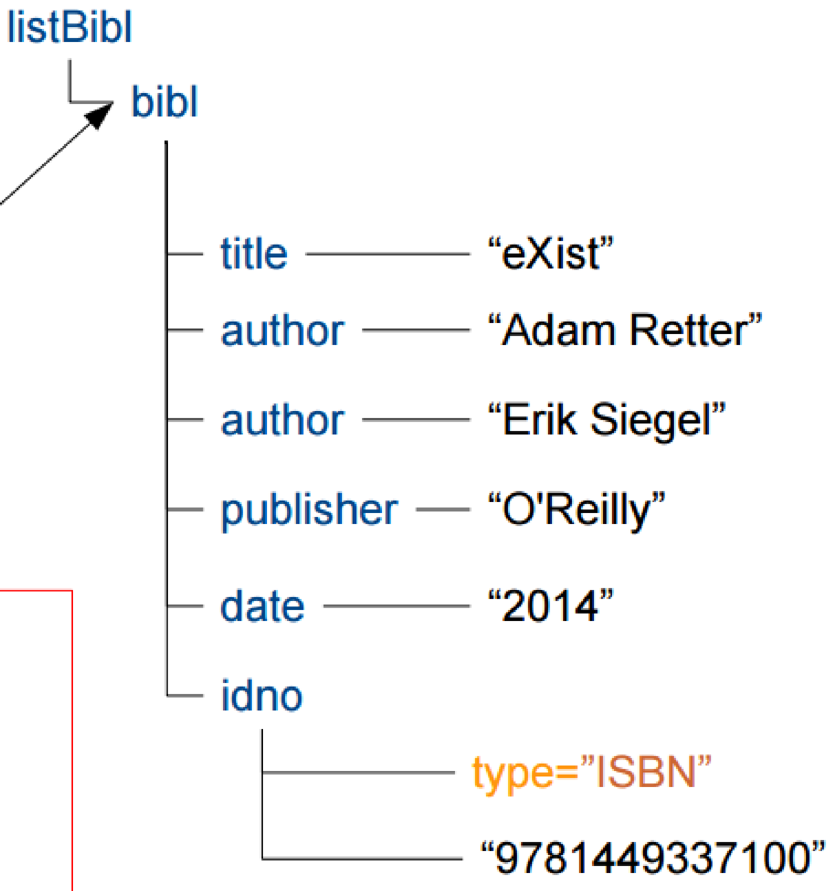
```
/listBibl/descendant-or-self::node()/author
```

```
<?xml version="1.0" encoding="UTF-8"?>
<listBibl>
  <bibl>
    <title>eXist</title>
    <author>Adam Retter</author>
    <author>Erik Siegel</author>
    <publisher>O'Reilly</publisher>
    <date>2014</date>
    <idno type="ISBN">9781449337100</idno>
  </bibl>
</listBibl>
```



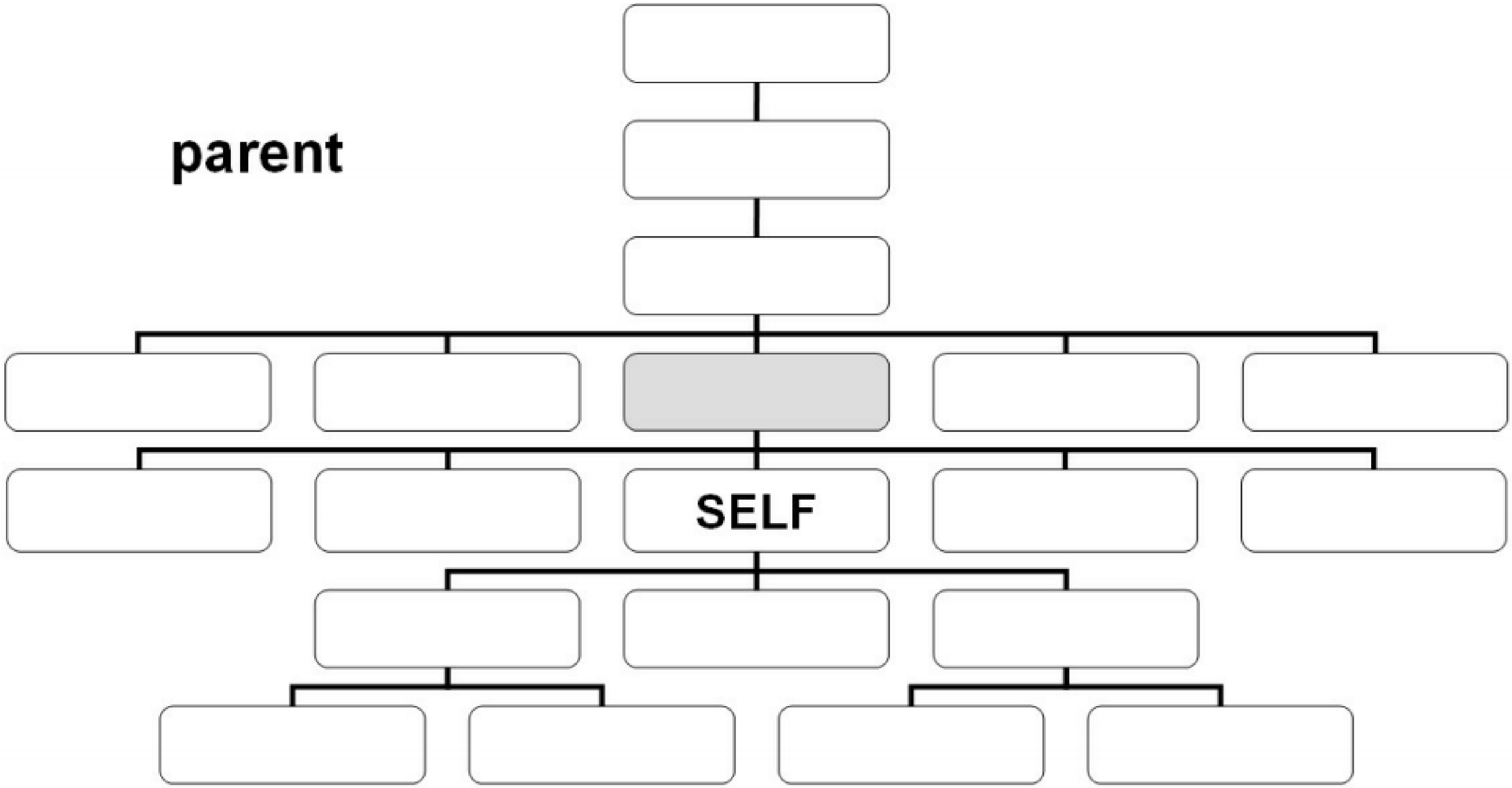
```
//bibl/self::bibl

Shortcut:

//bibl/.
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<listBibl>
  <bibl>
    <title>eXist</title>
    <author>Adam Retter</author>
    <author>Erik Siegel</author>
    <publisher>O'Reilly</publisher>
    <date>2014</date>
    <idno type="ISBN">9781449337100</idno>
  </bibl>
</listBibl>
```

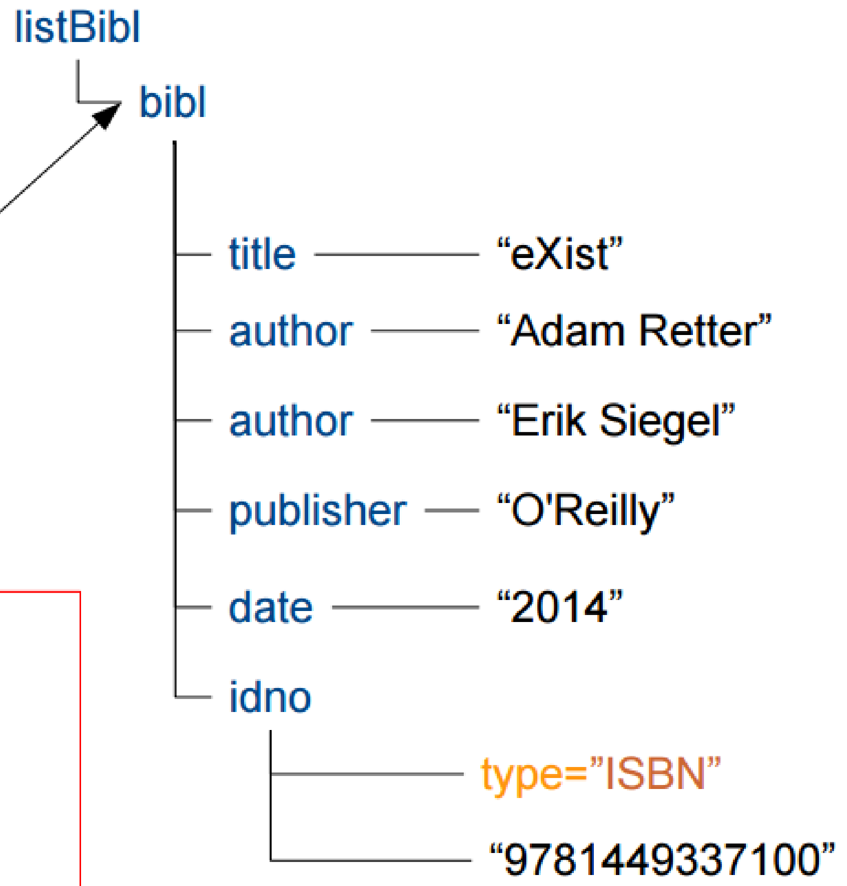


existsolutions

The self axis is frequently used inside predicates: //bibl/title[. = “eXist”] //bibl[./title = “eXist”] (unnecessary in this case) //listBibl[.//title = “eXist”] parent

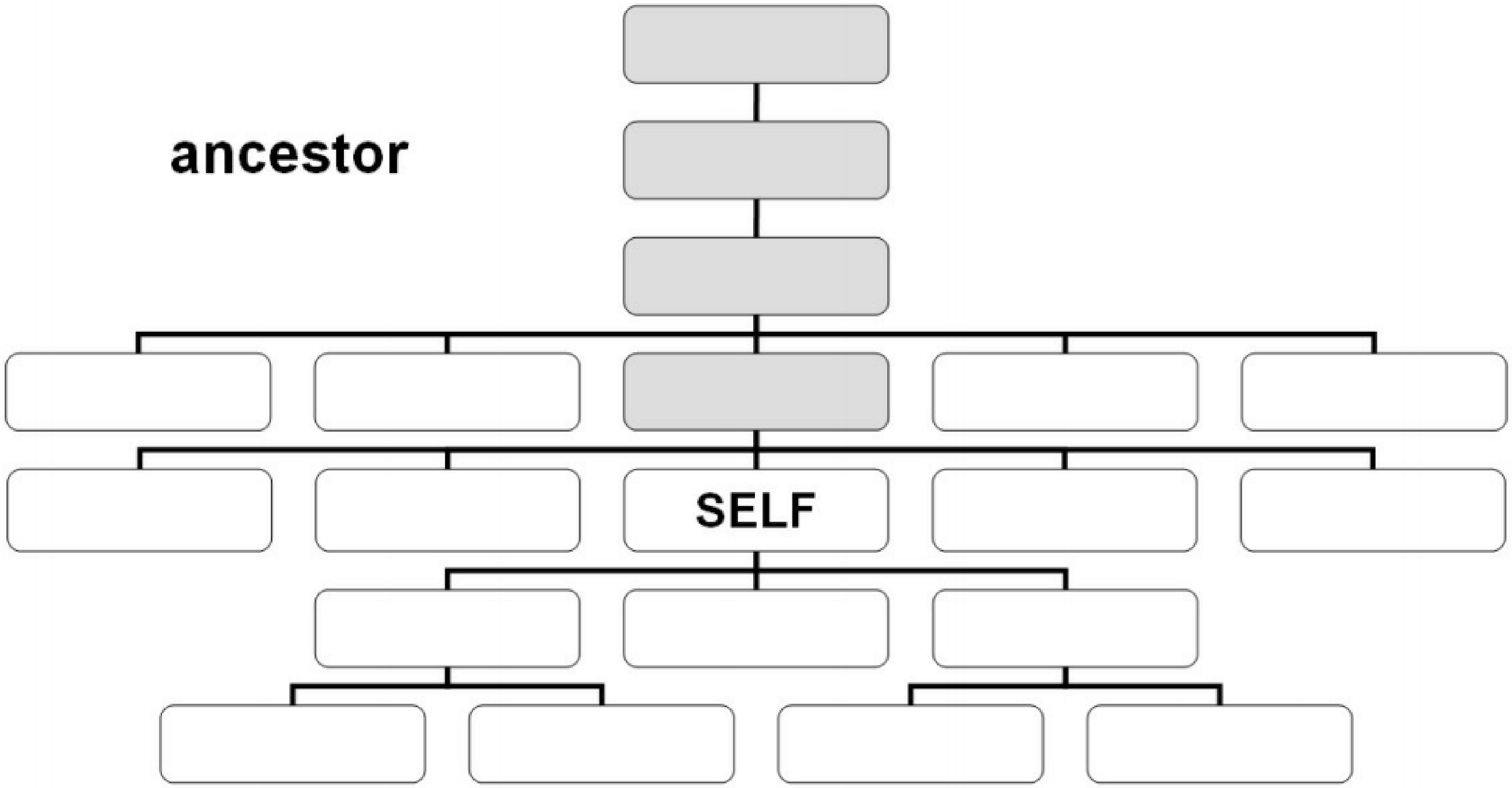


```
//title[. = "eXist"]/parent::bibl

Shortcut:

//title[. = "eXist"]/..
```

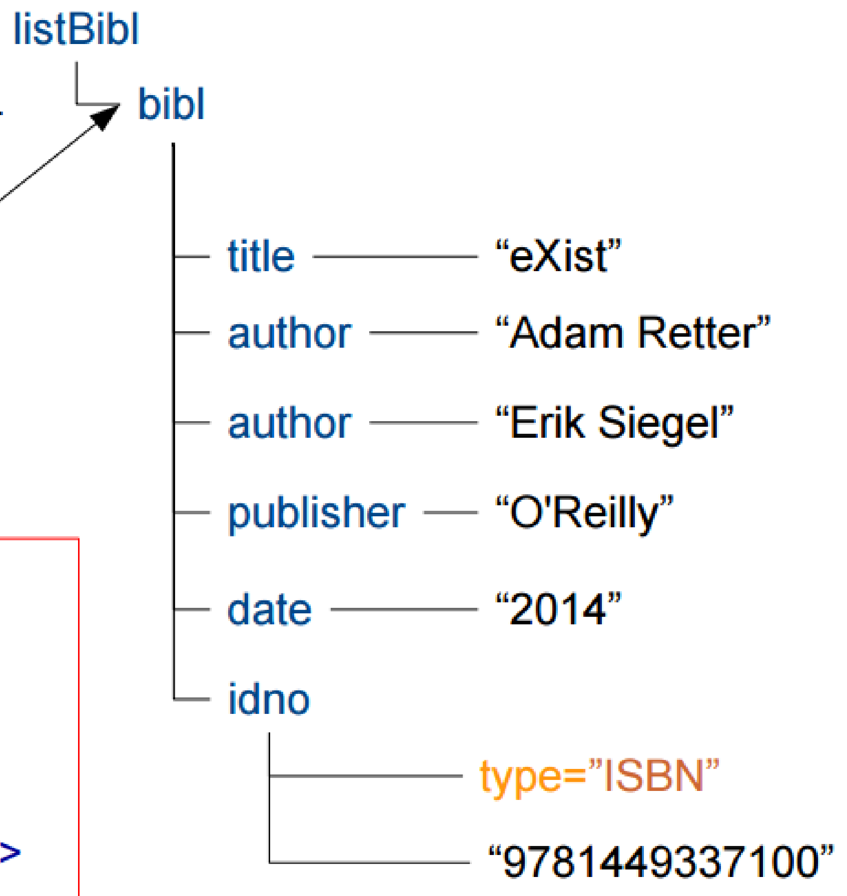


```xml
<?xml version="1.0" encoding="UTF-8"?>
<listBibl>
    <bibl>
        <title>eXist</title>
        <author>Adam Retter</author>
        <author>Erik Siegel</author>
        <publisher>O'Reilly</publisher>
        <date>2014</date>
        <idno type="ISBN">9781449337100</idno>
    </bibl>
</listBibl>
```

## ancestor



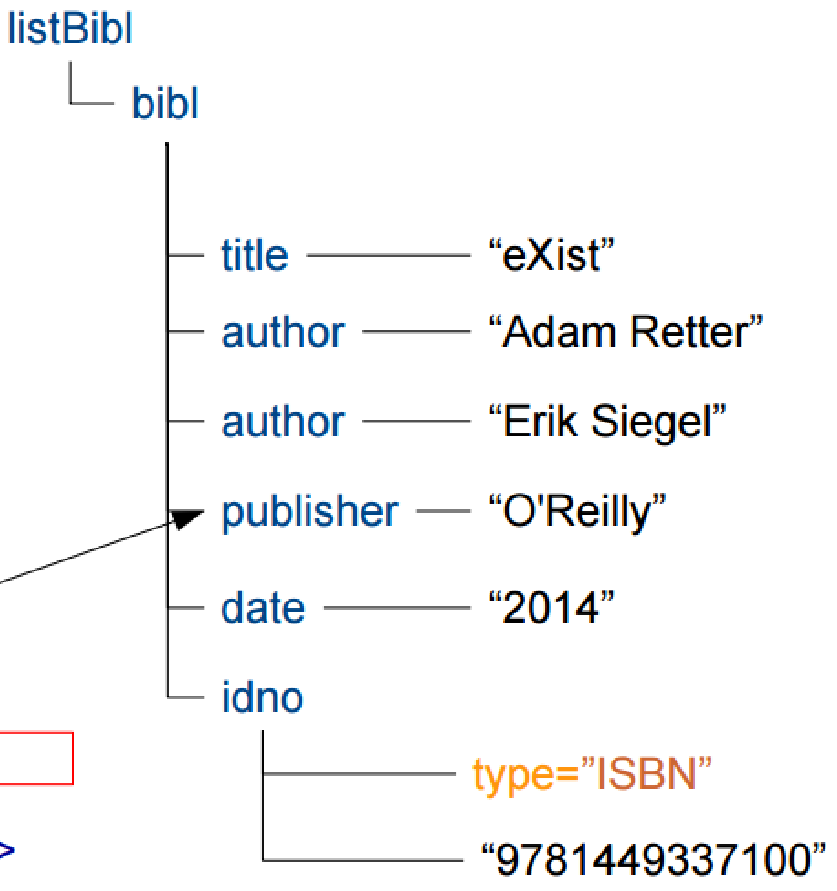
```
//title[. = “eXist”]/ancestor::bibl
```



```xml
<?xml version="1.0" encoding="UTF-8"?>
<listBibl>
    <bibl>
        <title>eXist</title>
        <author>Adam Retter</author>
        <author>Erik Siegel</author>
        <publisher>O'Reilly</publisher>
        <date>2014</date>
        <idno type="ISBN">9781449337100</idno>
    </bibl>
</listBibl>
```

```
//title[. = “eXist”]/
    following-sibling::publisher
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<listBibl>
   <bibl>
      <title>eXist</title>
      <author>Adam Retter</author>
      <author>Erik Siegel</author>
      <publisher>O'Reilly</publisher>
      <date>2014</date>
      <idno type="ISBN">9781449337100</idno>
   </bibl>
</listBibl>
```



| Shortcut | Axis | Explanation |
|---|---|---|
| . | self:: | the current context node |
| | child:: | child node - the default if no axis is given |
| | descendant:: | all descendant nodes |
| | descendant-or-self:: | descendants including the context node |
| @ | attribute:: | attribute node |
| | following-sibling:: | siblings following the context node |
| | preceding-sibling:: | sibling nodes before the context node |
| .. | parent:: | the parent node of the context node |
| | ancestor:: | all ancestor nodes of the context node |
| | following:: | nodes occurring after context node in document order |
| | preceding:: | nodes occurring before context node in document order |

## FUNCTIONS

```
contains(“hello world!”, “world”)
distinct-values((“One”, “Two”, “One”))
format-date(current-date(), "[MNn] [D1o], [Y0000]")
("Herbert", "Gustav", "Heinz")[starts-with(., "He")]
```

Default functions are normally used with empty namespace, e.g. contains() count() but sometimes you'll see them written with the predefined prefix fn:

| Function | Meaning |
|---|---|
| collection(“/db/apps/one”) | Selects one collection in /db/apps |
| doc(“/db/apps/one/this.xml”) | Selects one specified xml document |
| count(//node) | Counts the occurrence of a specified node in a given context” |
| distinct-values(//node) | Returns all distinct string values |
| contains(“hello”, “ll”) | Checks if the second string is contained within the first one, returns true or false |
| //node/@attr/string() | Returns the string value of the context node |
| starts-with($arg1, $arg2) | Checks if one string starts with another string |
| string-length($arg) | Returns the length of a string |
| current-date() | Returns the current date |

## EXPRESSION CONTEXT

The context sequence of an XPath expression is either

- Implicit!
- Explicitly specified with fn:doc() or fn:collection()

How a collection is defined depends on implementation. In eXist, fn:doc and fn:collection take a path pointing into the database:

- fn:doc("/db/apps/shakespeare/data/ham.xml")
- fn:collection(“/db/apps/shakespeare/data”)

## COLLECTION() & DOC() EXAMPLES

```
doc("/db/apps/shakespeare/data/ham.xml")//tei:sp[contains(tei:l, "kir
collection("/db/apps/shakespeare/data")//tei:sp[contains(tei:l, "king
count(doc("/db/apps/shakespeare/data/ham.xml")//tei:sp[contains(tei:l
count(collection("/db/apps/shakespeare/data")//tei:sp[contains(tei:l,
```

## COMMON PITFALLS

- Typos in element and attribute path steps! Writing //tei:spaeker instead of //tei:speaker will not cause an error, but simply returns nothing
- Namespaces! Using //speaker instead of //tei:speaker will not find anything
- Always check your XPath axes: //tei:div/tei:speaker returns nothing (correct: //tei:div//tei:speaker); similar: //tei:div/tei:speaker/tei:l instead of //tei:div/tei:speaker/tei:following-sibling::tei:l
- Build complex queries in small iterations. Split tasks into smaller steps and solve them one by one