FREE eBook

LEARNING

xml

Free unaffiliated eBook created from
Stack Overflow contributors.

#xml

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: xml

It is an unofficial and free xml ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official xml.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with xml

Remarks

XML is a markup language used to store hierarchical data in text files. It is also known as semi-structured data, like JSON. XML is machine-readable, yet can also be read and produced by people.

XML is made up of elements, sometimes casually referred to as a *tag soup*, which can themselves contain other elements and/or text. Elements may also contain attributes.

XML is often used for data exchange between platforms, especially over the internet. It is also increasingly used for storing semi-structured data in NoSQL data stores (XML databases/document stores). Furthermore, it has the flexibility to handle document-oriented data (text with markup), which makes it very popular in the publishing industry. XML is also widely used for configuration files.

One of the main reasons why XML is so widely used is that it is standardized, with many parsers available, including open source. This makes the cost of using XML lower than the invention of one's own new syntax.

More information about XML's origin and goals can be found in the official W3C Recommendation.

There are two versions of XML, shown in the table below. The editions of each version are just revisions of the original documents and not changes of the standards.

The first version of XML is 1.0. XML 1.1 was primary changed due to the Unicode version change from 2.0 to 3.1 and specifies a set of new rules for the use and interpretation of new Unicode characters.

Versions

Version	Release Date
1.0	1998-02-10
1.1	2001-12-13

Examples

Installation or Setup

XML is a syntax, which means a simple text editor is enough to get started.

However, having an XML-specific editor that shows you when and where your document is not

well-formed is almost indispensable for productivity. Such editors may also allow you to validate XML documents against an XML Schema, or even generate XML Schemas from XML documents (and vice versa).

Some examples of editors are oXygen, Atom, Eclipse and Altova XMLSpy. An alternate solution is to use a command-line XML parser such as Apache Xerces.

The basic building blocks

XML is made of basic building blocks, which are:

- element
- text
- attributes
- comments
- processing instructions

An element has angle brackets:

```
<element/>

<element>some content</element>
```

An attribute appears in an opening element tag:

```
<element
  attribute-name="attribute value"
  other-attribute='single-quoted value'>
  ...
</element>
```

Text can appear anywhere within or between elements:

```
<element>some more <b>bold</b> text</element>
```

Comments use the following syntax. It is important to know that XML comments, unlike in programming languages, are part of the model, and will be visible to the application above the parser.

```
<!-- this is a comment -->
```

Processing instructions allow passing messages to the consuming application (e.g., how to display, or a stylesheet, etc). XML does not restrict the format of processing instructions.

```
<?target-application these are some instructions?>
```

More details on building blocks can be found in this topic

Well-formedness

An XML document is a text file that conforms to the XML specification's well-formedness rules. Such a conforming document is said to be *well-formed* (not to be confused with *valid*). XML is very strict with well-formedness in comparison to other languages such as HTML. A text file that is not well-formed is not considered XML and cannot be used by consuming applications at all.

Here are some rules that apply to XML documents:

1. XML uses a much self-describing syntax. A prolog defines the XML version and the character encoding:

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

2. There must be exactly one top-level element.

   However, comments, processing instructions, as well as the initial XML declaration, are allowed at the top-level as well. Text and attributes are not.

   ```
   <?xml version="1.0"?>
   <!-- some comments -->
   <?app a processing instruction?>
   <root/>
   <!-- some more comments -->
   ```

3. Elements may nest, but must be "properly nested":

   ```
   <name>
     <first-name>John</first-name>
     <last-name>Doe</last-name>
   </name>
   ```

   The start and end tags of an embedded element have to be within the start and end tags of its container element. An overlapping of elements is illegal. In particular, this is not well-formed XML: `<foo><bar></foo></bar>`

4. Attributes may only appear in opening element tags or empty element tags, not in closing element tags. If attribute syntax appears between elements, it has no meaning and is parsed as text.

   ```
   <person first-name="John" last-name="Doe"/>
   ```

   This is not well-formed: ~~`<person></person first-name="John"/>`~~

5. Comments, processing instructions, text and further elements can appear anywhere inside an element (i.e., between its opening and closing tag) but not inside the tags.

   ```
   <element>
       This is some <b>bold</b> text.
   ```

```
    <!-- the b tag has no particular meaning in XML -->
</element>
```

This example is not well-formed: ~~<element <-- comment --> />~~

6. The `<` character may not appear in text, or in attribute values.
7. The `"` character may not appear in attribute values that are quoted with `"`. The `'` character may not appear in attribute values that are quoted with `'`.
8. The sequence of characters `--` may not appear in a comment.
9. Literal `<` and `&` characters must be escaped by their respective entities `<` and `&`.

Hello World

```
<?xml version="1.0"?>
<?speech-generator voice="Siri"?>
<root xmlns:vocabulary="http://www.example.com/vocabulary">
  <!-- These are the standard greetings -->
  <vocabulary:greetings>
    <vocabulary:greeting xml:lang="en-US" type="informal">
      Hi!
    </vocabulary:greeting>
    <vocabulary:greeting xml:lang="en-US" type="intermediate">
      Hello!
    </vocabulary:greeting>
    <vocabulary:greeting xml:lang="en-US" type="formal">
      Good morning to <b>you</b>!
    </vocabulary:greeting>
  </vocabulary:greetings>
</root>
```

Namespaces

Element and attribute names live in namespaces that are URIs. Namespaces are bound to prefixes that are used in the actual element and attribute names, which are called QNames.

This document binds a namespace to the prefix `prefix` and defines a default namespace, bound with the absence of prefix.

```
<?xml version="1.0"?>
<document
    xmlns="http://www.example.com/default-namespace"
    xmlns:prefix="http://www.example.com/another-namespace">
  <prefix:element/>
</document>
```

More details on namespaces can be found in this topic

Read Getting started with xml online: https://riptutorial.com/xml/topic/882/getting-started-with-xml

Chapter 2: Building blocks

Examples

Elements

Elements come with angle brackets are the most prominent building block of XML.

Elements can either be empty, in which case they are made of an empty tag (notice the ending slash):

```
<an-empty-element/>
```

Or they can have content, in which case they have an opening tag (no slash) and a closing tag (beginning slash):

```
<a-non-empty-element>Content</a-non-empty-element>
```

Elements can nest (but only between opening and closing tags):

```
<parent-element>
  <child-element/>
  <another-child-element>
    Some more content.
  </another-child-element>
</parent-element>
```

Element names are called QNames (qualified names). All above elements are in no namespace, but element names can also be defined in namespaces using prefixes like so:

```
<my-namespace:parent-element xmlns:my-namespace="http://www.example.com/">
  <my-namespace:child-element/>
  <my-namespace:another-child-element>
    Some more content.
  </my-namespace:another-child-element>
</my-namespace:parent-element>
```

Namespaces and element names are described in greater details in this section of the documentation.

Attributes

Attributes are name-value pairs associated with an element.

They are represented by values in single or double quotes inside the opening element tag, or the empty element tag if it is an empty element.

```
<document>
  <anElement foo="bar" abc='xyz'><!-- some content --></anElement>
  <anotherElement a="1"/>
</document>
```

Attributes are not ordered (unlike elements). The following two elements have the same sets of attributes:

```
<foo alpha="1" beta="2"/>

<foo beta="2" alpha="1"/>
```

Attributes cannot be repeated in the same element (unlike elements). The following document is not well-formed: ~~`<foo a="x" a="y"/>`~~ because the attribute `a` appears twice in the same element.

The following document is well-formed. Values can be identical, it is the attribute name that can't be repeated.

```
<foo a="x" b="x"/>
```

Attributes cannot be nested (unlike elements).

Text

Text is made of all characters outside of any markup (opening element tags, closing element tags, etc).

```
<?xml version="1.0"?>
<document>
  This is some text and <b>this is some more text</b>.
</document>
```

The precise XML terminology for text is *character data*. The XML specification actually uses the word *text* for the entire XML document, or a parsed entity, because it defines XML at the syntactic level. However some data models such as the XDM (XQuery and XPath Data Model), which represent XML documents as trees, refer to character data as *text nodes*, such that *text* is often understood as a synonym for character data in practice.

Character data may not contain a `<` character -- this would be interpreted as the first character of an opening element tag -- neither can it contain the `]]>` character sequence. The appropriate characters must be escaped with an entity reference instead.

```
<?xml version="1.0"?>
<document>
  It is fine to escape the &lt; character, as well as ]]&gt;.
</document>
```

For convenience, one can also escape a bigger chunk of text with a CDATA section (but the sequence `]]>` is still not allowed for obvious reasons):

```
<?xml version="1.0"?>
<document>
  <![CDATA[
    In a CDATA section, it is fine to write < or even & and entity references
    such as &amp; are not resolved.
  ]]>
</document>
```

Comments

Comments in XML look like so:

```
<!-- This is a comment -->
```

They can appear in element content or top-level:

```
<?xml version="1.0"?>
<!-- a comment at the top-level -->
<document>
  <!-- a comment inside the document -->
</document>
```

Comments cannot appear inside tags or inside attributes:

~~<element <!-- comment with --- inside --> />~~

or

~~<element attr="<!-- comment with --- inside -->"/>~~

are not well-formed.

The character sequence `--` cannot appear in the middle of a comment. This is not well-formed XML:

~~<!-- comment with --- inside -->~~

Comments in XML, unlike in other languages such as C++, are **part of the data model**: they are parsed, forwarded, and visible to the consuming application.

Processing instructions

A processing instruction is used to directly pass on some information or instruction to the application via the parser.

```
<?my-application some instructions ?>
```

The token after the initial question mark (here `my-application`) is called the target and identifies the application at which the instruction is aimed. What follows it is not further specified and it is up to the application to interpret it. Entity and character references are not recognized.

It can appear at the top-level, or in element content.

Read Building blocks online: https://riptutorial.com/xml/topic/1590/building-blocks

Chapter 3: DTD

Introduction

XML Document Type Declaration commonly known as DTD is a way to describe precisely the XML language. DTDs check the validity of, structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language. A DTD defines the structure and the legal elements and attributes of an XML document.

Examples

Document Type Declaration

An XML document can contain a DTD. DTD stands for *Document Type Declaration*. A DTD begins with `<!DOCTYPE root-element-name >` where `doc-element-name` must match the name of the so-called document element (the one element at the top-level).

```
<?xml version="1.0"?>
<!DOCTYPE document>
<document>
  <!-- the rest of the document -->
</document>
```

Entities

A DTD can contain entity declarations.

```
<?xml version="1.0"?>
<!DOCTYPE document [
  <!ENTITY my-entity "This is the replacement text">
]>
<document>
  <!-- the rest of the document -->
</document>
```

Entities are described in details in this topic.

XML document with an internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To reference it as internal DTD, standalone attribute in XML declaration must be set to yes.

An XML that describes a note that contains property to, from and message along with internal DTD will look like:

```
 <?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE note [
```

```
<!ELEMENT note (to,from,message>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT message (#PCDATA)>
]>
<note>
<to>Mr.X</to>
<from>Mr.Y</from>
<message>Stack Overflow is awesome </message>
</note>
```

XML document with an external DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL. To reference it as external DTD, the standalone attribute in the XML declaration must be set as no.

An XML that describes a note that contains property to, from and the message is given below.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Mr.X</to>
  <from>Mr.Y</from>
  <message>Stack Overflow is awesome</message>
</note>
```

External DTD for the above XML, *note.dtd* is given below

```
<!DOCTYPE note [
<!ELEMENT note (to,from,message>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT message (#PCDATA)>
]>
```

Read DTD online: https://riptutorial.com/xml/topic/3897/dtd

Chapter 4: Entities

Remarks

From a storage perspective, an XML document is made of entities. One of the entities is the document entity, which is the main XML document itself.

Entities can be classified like so (tentatively sorted by descending order of usage):

- **document entity**: this is the main XML file.
- **internal general entities**: this is the most common one besides the document entity, and the one most XML users are aware of. Often, the word **entity** is casually used for them. They allow specifying some shortcuts for longer replacement texts in document content. They are declared in the DTD.
- **the external DTD subset**: another file in which part of the DTD is outsourced.
- **parameter entities**: shortcuts, for use in the DTD.
- **external parsed general entities**: they are XML fragments stored in other files.
- **unparsed entities**: these can be any files on which XML places no restrictions, including images, sounds, etc.

In many cases, an XML document consists solely of the document entity.

Examples

Pre-defined general entities

XML pre-defines five general entities that can be used without declaring them:

```
& " ' < >
```

They are associated with the names `amp`, `quot`, `apos`, `lt` and `gt`.

```
<?xml version="1.0"?>
<entities>
  &amp; is an ampersand.
  &quot; is a quote.
  &apos; is an apostrophe.
  &lt; is a lower-than sign.
  &gt; is a greater-than sign.
</entities>
```

User-defined general (internal) entities

It is possible to define one's own general entities. The declaration occurs in the DTD subset, with a name and the associated replacement text.

It can then be used in the document using the entity reference syntax `&...;`, either in text, or in

attribute values.

```
<?xml version="1.0"?>
<!DOCTYPE my-document [
  <!ENTITY my-entity "This is my entity">
]>
<my-document>
    The entity was declared as follows: &my-entity;
    <element attribute="Entity: &my-entity;"/>
</my-document>
```

External parsed entities

XML fragments, also known under the name of *external parsed entities*, can be stored in separate files.

XML fragments, unlike XML documents, are less restrictive, in that several elements can appear top-level, as well as text nodes. Like an XML document, an external parsed entity may begin with an XML declaration, but this declaration is not considered part of its replacement text.

This is an example of external parsed entity:

```
<?xml version="1.0" encoding="UTF-8"?>
This is some text
<element/>
<element/>
```

An external parsed entity can then be declared in an XML document, in the DTD, and it can be used with an entity reference, which has the same syntax as for general internal entities:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
<!ENTITY fragment SYSTEM "fragment.xml">
]>
<root>
    &fragment;
</root>
```

With the entity reference resolved, this document is equivalent to:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
<!ENTITY fragment SYSTEM "fragment.xml">
]>
<root>
  This is some text
  <element/>
  <element/>
</root>
```

Every opening element tag in an external parsed entity must have a corresponding ending tag: it is not allowed to spread single elements over multiple entities, nor to spread markup.

A validating parser is required to resolve the entity reference and include its replacement text in the document as above. A non-validating parser may skip this, and instead tell the consuming application that there is an unresolved reference to an external parsed entity.

Read Entities online: https://riptutorial.com/xml/topic/2302/entities

Chapter 5: Escaping

Remarks

Characters can be escaped in XML using entity references and character references, or CDATA sections.

XML pre-defines five entities:

Named entity	Replacement text
amp	&
quot	"
apos	'
lt	<
gt	>

Consuming applications will not know whether each character has been escaped or not, and how.

Examples

Ampersand

The `&` character appears first in entity references and must be escaped in element content or in attribute content.

```
<?xml version="1.0"?>
<document attribute="An ampersand is escaped as &amp;">
  An ampersand can also be escaped as &amp; in element content.
</document>
```

Lower-than sign

The `<` character appears first in entity tags and must be escaped in element content or in attribute content.

```
<?xml version="1.0"?>
<document attribute="A lower-than sign is escaped as &lt;">
  2 + 2 &lt; 5
</document>
```

Greater-than sign

The `]]>` character sequence is not allowed in element content. The easiest way to escape it is to escape `>` as `>`.

```
<?xml version="1.0"?>
<document>
  The sequence ]]&gt; cannot appear in element content.
</document>
```

Apostrophes and quotes

Attribute values can appear in simple or double quotes. The appropriate character must be escaped.

```
<?xml version="1.0"?>
<document
  quot-attribute="This is a &quot;double quote&quot; and this one is 'simple'"
  apos-attribute='This is a &apos;simple quote&apos; and this one is "double"'>
</document>
```

CDATA sections

Longer portions of text containing special characters can be escaped with a CDATA section. CDATA sections can only appear in element content.

```
<?xml version="1.0"?>
<document>
  This is a CDATA section : <![CDATA[ plenty of special characters like & < > " ; ]]>
</document>
```

A CDATA section cannot contain the sequence `]]>` because it ends it.

Character references

Characters can be escaped using character references, in element content or attribute values. Their Unicode codepoint can be specified in decimal or hex.

```
<?xml version="1.0"?>
<document>
  The line feed character can be escaped with a decimal (&#10;) or hex (&#xA;)
  representation of its Unicode codepoint (10).
</document>
```

XML restricts characters that can appear in a document, even escaped. In particular, the only control characters allowed are line feed (10), carriage return (13) or horizontal tab (9).

Read Escaping online: https://riptutorial.com/xml/topic/3685/escaping

Chapter 6: Namespaces

Remarks

Element and attribute names in XML are called QNames (qualified names).

A QName is made of:

- a namespace (a URI)
- a prefix (an NCName, NC because it contains no colon)
- a local name (an NCName)

Only the namespace and the local name are relevant for comparing two QNames. The prefix is only a proxy to the namespace.

The namespace and prefix are optional, but the namespace is always present if the prefix is present (this is ensured at the syntactic level, so this cannot be done wrong).

The lexical representation of a QName is `prefix:local-name`. The namespace is bound separately using the special `xmlns:...` attributes (reminder: attributes beginning with *xml* are reserved in XML).

If the prefix is empty, no colon is used in the lexical representation of the QName, which only contains the `local-name`. QNames with an empty prefix either have no namespace (if no default namespace is in scope) or are in the default namespace.

Examples

Bind a prefix to a namespace

A namespace is a URI, but to avoid verbosity, prefixes are used as a proxy.

In the following example, the prefix `my-prefix` is bound to the namespace `http://www.example.com/my-namespace` by using the special attribute `xmlns:my-prefix` (`my-prefix` can be replaced with any other prefix):

```
<?xml version="1.0"?>
<my-prefix:foo xmlns:my-prefix="http://www.example.com/my-namespace">
  <!-- the element my-prefix:foo
       lives in the namespace http://www.example.com/my-namespace -->
</my-prefix:foo>
```

Absence of namespace

In XML, element and attribute names live in namespaces.

By default, they are in no namespace:

```
<?xml version="1.0"?>
<foo attr="value">
  <!-- the foo element is in no namespace, neither is the attr attribute -->
</foo>
```

Irrelevance of prefixes

These two documents are semantically equivalement, as namespaces matter, not prefixes.

```
<?xml version="1.0"?>
<myns:foo xmlns:myns="http://www.example.com/my-namespace">
</myns:foo>

<?xml version="1.0"?>
<ns:foo xmlns:ns="http://www.example.com/my-namespace">
</ns:foo>
```

Default namespace

The default namespace is the namespace corresponding to the absence of any prefix. It can be declared with the special `xmlns` attribute.

```
<?xml version="1.0"?>
<foo xmlns="http://www.example.com/my-namespace">
  <!-- the element foo is in the namespace
       http://www.example.com/my-namespace -->
</foo>
```

If no default namespace is declared, then names with no prefix are in no namespace.

Attribute names with no prefix

Elements and attributes behave differently with respect to default namespaces. This is often the source of confusion.

An attribute whose name has no prefix lives in no namespace, **also when a default namespace is in scope**.

```
<?xml version="1.0"?>
<foo attr="value" xmlns="http://www.example.com/my-namespace">
  <!-- The attribute attr is in no namespace, even though
       a default namespace is in scope. The element foo,
       however, is in the default namespace. -->
</foo>
```

Scope of namespace bindings

A namespace binding (special `xmlns` or `xmlns:...` attribute) is in scope for all the descendants of the enclosing element, including this element.

```
<?xml version="1.0"?>
<root>
  <my:element xmlns:my="http://www.example.com/ns1">
    <!-- here, the prefix my is bound to http://www.example.com/ns1 -->
  </my:element>
  <my:element xmlns:my="http://www.example.com/ns2">
    <!-- here, the prefix my is bound to http://www.example.com/ns2 -->
  </my:element>
</root>
```

The binding can be overriden in a nested element (this affects readability though):

```
<?xml version="1.0"?>
<my:element xmlns:my="http://www.example.com/ns1">
  <!-- here, the prefix my is bound to http://www.example.com/ns1 -->
  <my:first-child-element/>

  <my:child-element xmlns:my="http://www.example.com/ns2">
    <!-- here, the prefix my is bound to http://www.example.com/ns2,
         including for the element my:child-element -->
  </my:child-element>

  <!-- here, the prefix my is bound to http://www.example.com/ns1 -->
  <my:last-child-element/>

</my:element>
```

It is very common to declare all namespace bindings in the root element, which improves readability.

```
<?xml version="1.0"?>
<root
  xmlns="http://www.example.com/default-namespace"
  xmlns:ns1="http://www.example.com/ns1"
  xmlns:ns2="http://www.example.com/ns2">

  <ns1:element>
    <ns2:other-element/>
  </ns1:element>

</root>
```

Read Namespaces online: https://riptutorial.com/xml/topic/1593/namespaces

Chapter 7: XML Catalogs

Introduction

An XML catalog is made up of entries from one or more catalog entry files. A catalog entry file is an XML file whose document element is catalog and whose content follows the XML catalog DTD defined by OASIS at http://www.oasis-open.org/committees/entity/spec.html. Most of the elements are catalog entries, each of which serves to map an identifier or URL to another location.

Examples

Catalog entry to resolve DTD location

1

2 3

```
<public
   publicId="-//OASIS//DTD DocBook XML V4.5//EN"  4
   uri="docbook45/docbookx.dtd"/>

<system
   systemId="http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd"  5
   uri="docbook45/docbookx.dtd"/>

<system
   systemId="docbook4.5.dtd"    6
   uri="docbook45/docbookx.dtd"/>
```

Read XML Catalogs online: https://riptutorial.com/xml/topic/10875/xml-catalogs

Chapter 8: XML Schema

Introduction

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types.

Examples

An Example of XSD Document

An XSD that describe a contact information about a company is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://NamespaceTest.com/CommonTypes"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  elementFormDefault="qualified">
<xs:element name="contact">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string" />
            <xs:element name="company" type="xs:string" />
            <xs:element name="phone" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

In the above example the attributes in second line

> <xs:schema targetNamespace="http://NamespaceTest.com/CommonTypes"
> xmlns:xs="http://www.w3.org/2001/XMLSchema"
> elementFormDefault="qualified">

Attributes 'targetnamespace' and elementFormDefault are optional.

Read XML Schema online: https://riptutorial.com/xml/topic/8983/xml-schema

Credits

S. No	Chapters	Contributors
1	Getting started with xml	Al.G., Burkart, Community, Elizaveta Revyakina, Ghislain Fourny, Joe, JohnRC, Kin, MAZux, Mohammad Arman, RamenChef, TuringTux, w5m, Wolfgang Schindler
2	Building blocks	Ghislain Fourny, Hoylen
3	DTD	Dipesh Poudel, Ghislain Fourny
4	Entities	Ghislain Fourny
5	Escaping	Ghislain Fourny
6	Namespaces	Ghislain Fourny
7	XML Catalogs	Mistletoe
8	XML Schema	Dipesh Poudel