1. The XML language

1.1. Introduction

Learning objectives, concepts, and skills:

- Identify the areas of application of XML language and its advantages.
- Identify the structure of an XML document and its syntactic rules.
- Create well-formed XML documents.
- Identify the advantages provided by using namespaces.
- Select and declare namespaces.
- Create documents that use XML namespaces.

Introduction to XML language

The main feature of a markup language is that it adds to the content of the text, **tags** that enrich it with additional information about its structure or presentation.

HTML is, surely, the most used markup language. A simple document that uses the HTML language could be, for example, the following:

```
<!DOCTYPE html>
<html>
 <head>
     <TITLE>Example of HTML</TITLE>
 </title>
 <body>
  <!-- Example.html -->
  <h1>HTML 1.0</h1>
  <p>The first version of HTML was developed by Tim Berners-Lee.</p>
 </body>
</html>
```

In HTML, there are several predefined tags (html, head, body, h1, p, ...). In early versions of HTML, tags were used to define both the structure of the document and its appearance, but that has changed with time so that currently it is advised to define the appearance using CSS style sheets and use the tags only to define the structure of the document.

The fundamental difference between the XML language and other languages like HTML is that XML does not have a set of predefined tags. The set of tags to be used is defined for each document, in a way that let to identify its content. For example, an XML document with information about individuals may contain:

```
<surname>Magán</surname>
<name>María</name>
<birthdate>04/08/1982</birthdate>
```

And an XML document on vocational modules:

```xml
<modules>
  <module>
    <name>Contornos de desenvolvemento</name>
    <num_hours>107</num_hours>
  </module>
  <module>
    <name>Linguaxes de marcas</name>
    <num_hours>133</num_hours>
  </module>
</modules>
```

That is, XML (eXtensible Markup Language) is not a markup language. XML is a **meta-language** we can use to define our own markup languages for each specific need. Using XML we can create languages that store the information that a specific application needs.

XML is a subset of the SGML (Standard Generalized Markup Language) language, created to simplify the creation of the grammar of markup languages. XML designers left out the least-used parts of the SGML, making the XML specification fill 30 pages, compared to the 500 pages of the SGML.

Advantages of using XML

The **XML** language was created to **structure**, **store,** and **transport** information. The main requirements that were considered in the creation of XML were:

- It was easy to create XML documents, and they could be easily read and understood.
- XML documents could be easily processed by a computer.
- It could serve in multiple fields while maintaining compatibility with SGML.

The main advantages derived from the use of XML documents are:

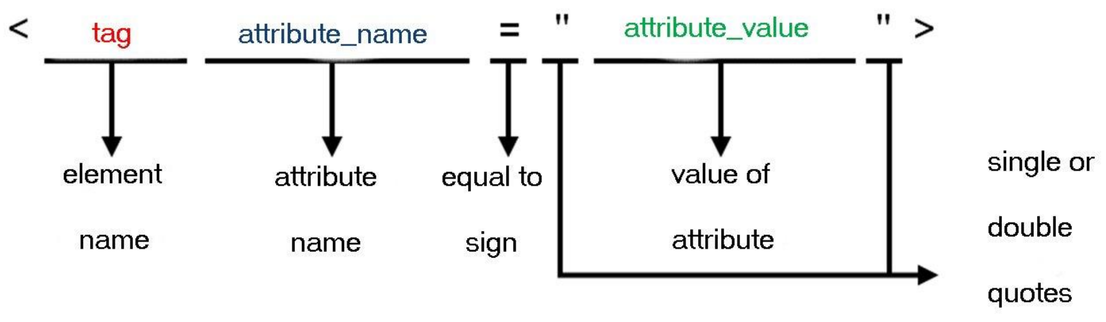
- The structure of an XML document can be understood without difficulty.
- XML provides structure and information about the data of the documents; using the elements of the language correctly, we can identify the parts of a document and its concrete meaning.
- XML facilitates the communication of information between applications, independently of the platform in which they are developed and of the data sources they use.
- XML documents can be easily extended; when creating an XML-based language, it is guaranteed that in the future new tags can be added to it while maintaining compatibility with the existing ones.

- There are standard XML document analysers that we can use to process any XML-based language. It is not necessary to create specific analysers for each language, which speeds up the development of applications that use XML documents.

Content of an XML document

Elements

The base of an XML document is the element. Each element is formed by an opening and a closing tag, using both characters "<" and ">" in both cases. The content of an element is everything that appears between the two tags.



An element may contain:

- **attributes**, that will always appear in the opening tag.
- **text**, between the opening tag and the closing tag.
- **other elements**.

For instance, the element "`module`" contains the attributes "`sessions`" and "`hours`", a text, and the sub-element "`teacher`":

```
<module sessions="5" hours="133">Linguaxes de marcas e sistemas de información
   <teacher>María Mato</teacher>
</module>
```

There may also be empty elements, which can be represented by two tags or by one:

```
<active></active>
<active/>
```

Names of the elements

The name of the element is the word that appears in the opening and closing tags, that must be the same.

The name of an element may contain any alphabetic character (stressed or not), numbers and some punctuation marks such as ".", "_" And "-", but may **not** contain **spaces**. Also, the first character cannot be a number or any of the characters "." or "-". For example, the following element names would not be valid:

```
<1January2013></1January2010>
<+element></+element>
<.element></.element>
<-element></-element>
```

Although, as reflected in the language specification, the ":" character can also be used in the names of the elements, it should be avoided as we will see later leads to confusion with the syntax used in namespaces.

The names of elements that start with "`xml`" characters are also reserved, in any combination of lower and upper case, such as "`Xml`", "`XmL`" ou "`xmL`".

Attributes

Elements may or may not contain attributes. Attributes are a way to incorporate features or properties to the elements. For example:

```
<module sessions="5" hours="133"></module>
```

When an element contains attributes, they must appear in the opening label separated from each other and from the name of the element by spaces.

There is no limit to the maximum number of attributes that an element can contain, and the order in which they are within the element is not relevant either, but a specific element cannot have attributes with a repeated name.

The names of the attributes must respect the same rules as the names of the elements, as for the types of characters allowed.

Attributes or subelements

Sometimes we must choose between using a subelement or an attribute to store certain information. For example, to refer to the teacher who imparts a module we could put:

```
<module>Linguaxes de marcas e sistemas de información
   <teacher>Marta Nogueira</teacher>
</module>
```

Or:

```
<module teacher="Marta Nogueira">Linguaxes de marcas e sistemas de información
</module>
```

To make the choice, we must remember that the attributes are designed to store information (descriptions) about the data and can only store a single value, while the elements can store multiple data and even extend it through nested structures of several elements.

In the previous example, only with the first solution we could add information about the teacher or indicate a second teacher for the same module.

Attribute values

The value of the attribute must be enclosed in quotation marks, either single or double, and separated from the attribute name by a "=" sign. Some attributes can use single quotes, and other attributes can use double quotes, but both types cannot be mixed in the same attribute. For example, the following attribute "`attrib3`" would not be valid:

```
<element attrib1="value1" attrib2='value2' attrib3="value3'></element>
```

In XML all the attributes of an element must have a defined value. We cannot have attributes in which its value is not indicated. For example, it would not be valid:

```
<element attrib1></element>
```

Instead, we should do something like:

```
<element attrib1="yes"></element>
```

Or:

```
<element attrib1=""></element>
```

Text

The content of an element between the opening and closing tags, can be text, other elements, or both.

The text of XML documents can contain any type of character except "<" and "&". Thus, it would not be valid:

```
<author>Stevie Ray Vaughan & Double Trouble</author>
```

The same restriction applies to attribute values, which in addition to the characters "<" and "&" cannot contain the same type of quotation that is used to delimit it. For example, it would not be valid:

```
<song name='Don't bang the drum'></song>
```

If we want to use one of these characters within the text or within the value of an attribute, we have two options:

- **Use a reference to a Unicode character**. References always start with "&#" (or "&#x" if we use hexadecimal) and end in a semicolon (;). For example, the characters "&" and "<" could be replaced respectively by the references "`&#38;`" and "`&#60;`" (in hexadecimal they would be "`&#x26;`" and "`&#x3C;`" respectively).
- **Use entities**. As we will see below, using entities we could replace the characters "&" and "<" respectively by the strings "`&amp;`" e "`&lt;`".

Thus, in the first example we could write:

```
<author>Stevie Ray Vaughan &#38; Double Trouble</author>
```

Or:

```
<author>Stevie Ray Vaughan &amp; Double Trouble</author>
```

Comments

Sometimes it is convenient to insert comments in an XML document. These will be ignored when document information is processed. They have the same format as in HTML language, that is, start with the string "<!--" and ending with "-->".

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

They may appear anywhere in the document where we could write text and may contain any character except the double-hyphen combination "--".

Processing instructions

Processing instructions are used to give information to applications that process XML documents. Although they can appear anywhere in a document, they usually are placed at the beginning. They are delimited by the character sets "<?" and "?>". For example:

```
<?xml-stylesheet type="text/xsl" href="plantilla.xsl"?>
```

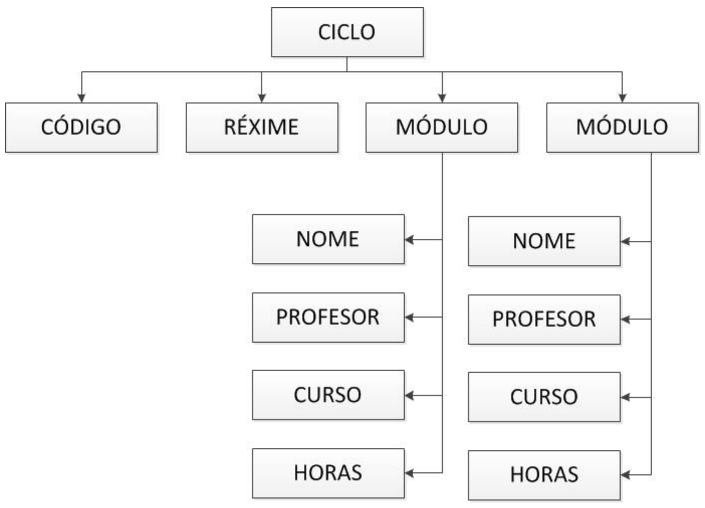
We will work with processing instructions when we learn about the XSLT language.

Well-formed XML documents

A document is called "**well-formed**" if it respects the rules we have seen up to now, and the following ones described in the official XML specification:

- All elements of the document are nested in the form of a tree, with only one root element, which contains the rest of the elements. For example:

```
<ciclo>
  <código>DAM</código>
  <réxime>Ordinario</réxime>
  <módulo>
    <nome>Linguaxes de marcas e sistemas de información</nome>
    <profesor>Marta Mato</profesor>
    <curso>1º</curso>
    <horas>133</horas>
  </módulo>
  <módulo>
    <nome>Contornos de desenvolvemento</nome>
    <profesor>Manuel Lamas</profesor>
    <curso>2º</curso>
    <horas>140</horas>
  </módulo>
</ciclo>
```



- All elements must be closed, and this should always be done in a structured way, that is, in reverse order of their opening: the last item to be opened will be the first to be closed.

```
<elementA>
  <elementB>
  </elementB>
</elementA
```

The closing tag of an element must match exactly the name that appears on the opening tag, including lowercase and uppercase.

In the 1.1 language specification, XML documents must begin with a prolog.

XML Specifications

There are two specifications of the XML language: 1.0 and 1.1. The differences between them are scarce.

The version 1.1 adds to the previous one:

- Complete Unicode support. Specification 1.0 is limited to the characters collected in Unicode 2.0, while the 1.1 specification covers even characters to be collected in future versions of Unicode.
- The ability to recognize other end-of-line characters. In XML 1.0 we could use "`&#xA;`" for the new line and "`&#xD;`" for the carriage return. Version 1.1 adds support for end-of-line characters defined by Unicode ("`&#x2028;`") and by EBCDIC ("`&#x85;`").
- The obligation to add a prolog to the documents to be considered well-formed.

XML 1.0 is the **most used version**.

Unless you have a specific requirement to work with XML 1.1 (which is very rare), you should use the XML 1.0 specification.

The prolog

The prolog of an XML document must appear at the beginning of it and must specify (in XML 1.1) the version of the language used. It has the format:

```
<?xml version="1.1"?>
```

Optionally, it can also include other attributes:

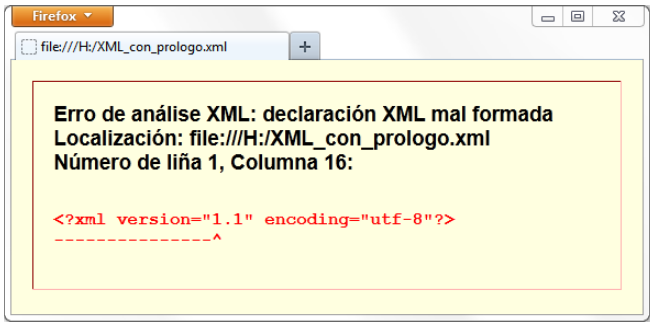
- **encoding**, to indicate the type of character encoding that is used in the document.
- **standalone**, which is used to indicate whether the document depends on other external documents or not to verify the validity of the grammar used in it.

All XML processors should be able to read `UTF-8` and `UTF-16` encoded documents and infer by themselves what one of them is used in a particular document; if this is the case, it is not necessary but advisable to use the attribute `encoding`. For other coding systems (such as `ISO-8859-x`), it must be verified that it is supported by the processor (the most common ones have no problems) and it will be mandatory to specify the `encoding` attribute in the prolog.

For example, a common prolog for an XML document is:

```
<?xml version="1.1" encoding="utf-8"?>
```

We should keep in mind that some browsers do not correctly process XML documents that contain a prolog. Some web browsers do not show the documents when the XML version indicated in the prolog is 1.1.



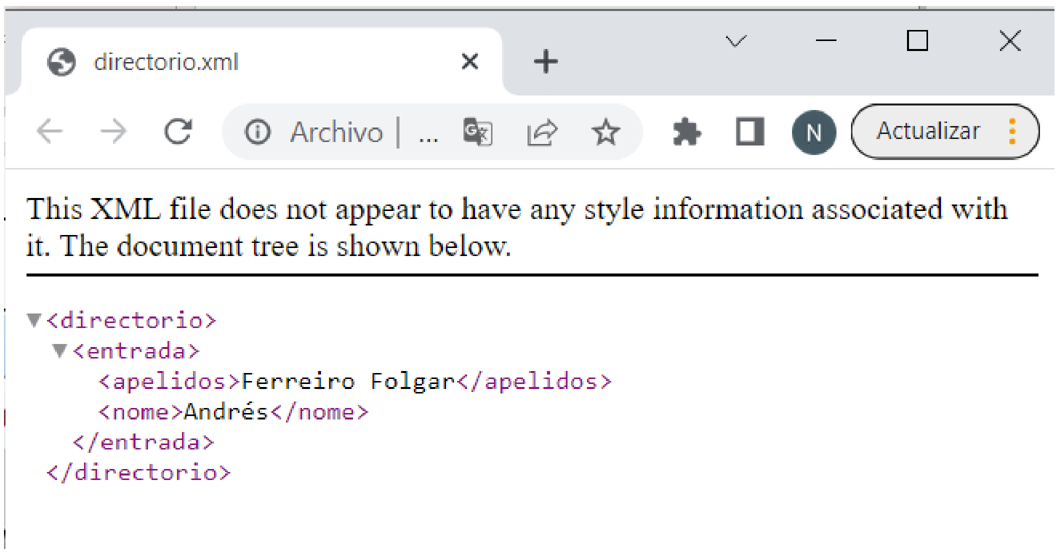
```
Erro de análise XML: declaración XML mal formada
Localización: file:///H:/XML_con_prologo.xml
Número de liña 1, Columna 16:

<?xml version="1.1" encoding="utf-8"?>
---------------^
```

Well-formed documents and web browsers

At present, browsers can distinguish well-formed documents from bad-formed ones. When we open an XML document in a browser, if the document is well formed, we will see the information it contains in the form of a tree with dynamic nodes. If the document has errors, the navigator will point out the first element that contains the error.

Let's see how a well-formed document is displayed in the browser:

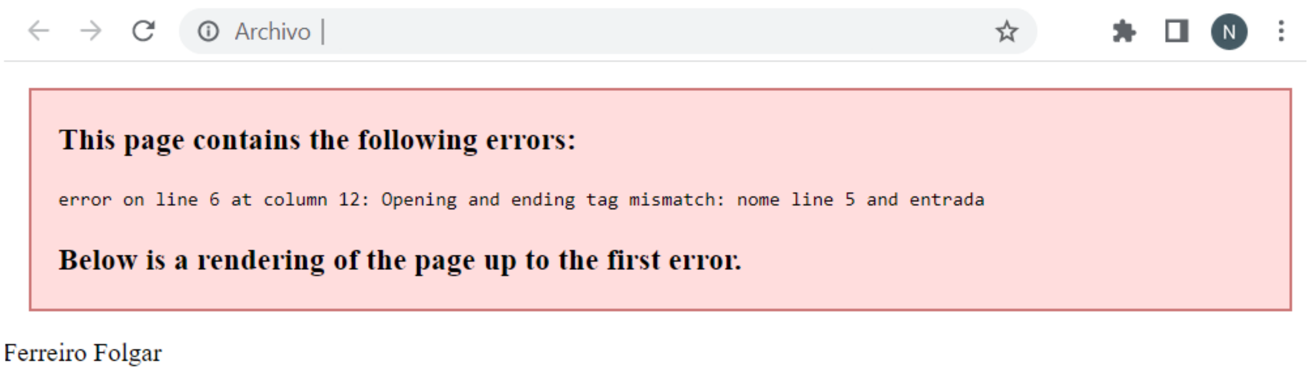
```
<?xml version="1.1" encoding="ISO-8859-1"?>
<directorio>
 <entrada>
   <apelidos>Ferreiro Folgar</apelidos>
   <nome>Andrés</nome>
 </entrada>
</directorio>
```



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<directorio>
  ▼<entrada>
     <apelidos>Ferreiro Folgar</apelidos>
     <nome>Andrés</nome>
   </entrada>
 </directorio>
```

If we modify the previous document by removing the closing tag </nome> we would get a bad-formed document:



This page contains the following errors:

```
error on line 6 at column 12: Opening and ending tag mismatch: nome line 5 and entrada
```

Below is a rendering of the page up to the first error.

Ferreiro Folgar

White space

XML defines a set of characters such as "whitespace", which processors allow to make the code more readable (for example, slide the labels to reflect the nesting). These characters are:

- Spaces (32).
- Tabs (9).
- Line breaks (10).

- Carriage returns (13).

Normally, the applications that process the XML documents eliminates the remaining spaces from their content. That is, the actual content of the element "apelidos" after being processed would be the same if we write:

```
<apelidos>Ferreiro Folgar</apelidos>
```

That if we write:

```
<apelidos>
   Ferreiro Folgar
</apelidos>
```

Content language identification

There is another special attribute defined in the XML specification that is used to identify the language in which the content of an item is written, "xml:lang". Its value is a two-letter code assigned in the ISO 639-1 standard (for instance, "gl" for galician or "es" for spanish).

An XML document that uses this attribute is:

```
<canción título="Automatic imperfection" xml:lang="en">
There are children playing on the manpost
'tending they are kids
They can climb up and fall down hard on their knees
…
</canción>
<canción título="Lo que sueñas vuela" xml:lang="es">
Caminando sin pensar,
despacito, sin tiempo que ganar
…
</canción>
```

Entities

Entities are XML structures with an associated name. To refer to an entity, the following syntax is used:

```
&entityName;
```

Referencing an entity by its name automatically inserts its content instead of its reference. We have already seen one before, and in the following unit we will discuss how to create entities, but for now we will see which entities are predefined in the XML specification:

- "&amp;" to reference the character "&".
- "&lt;" to reference the character "<".
- "&gt;" to reference the character ">".
- "&quot;" to reference the character double quote """.

- "`&apos;`" to reference the single quote "`'`" character.

The CDATA section

Sometimes the use of entities can be very uncomfortable and make difficult the reading of the document. That's why it can sometimes be very useful to use a section `<![CDATA[ ... ]]>`".

The CDATA section is opened with the characters "`<![CDATA[`"and closed with "`]]>`". The content inside a CDATA section is ignored by the analyser, so we can use characters like "`<`" and "&".

For example, if we wanted to add in an XML element some content written in HTML language, like the following:

```
<h1>Antony & The Johnsons</h1>
```

Using entities, we would have to write:

```
<author>&lt;h1&gt;Antony &amp; The Johnsons&lt;/h1&gt;</author>
```

By using a CDATA section, the content is clearer:

```
<author><![CDATA[<h1>Antony & The Johnsons</h1>]]></author>
```

The only logical limitation to its content is that within a CDATA section, the closing string "`]]>`" cannot appear. In this case, it should be replaced by "`]]&gt;`".

The need for namespaces

A namespace is a way of grouping elements and attributes with a common origin and to differentiate them from other elements with the same name.

For example, suppose that a food store creates an XML grammar for the products it sells, so that a typical XML document could be:

```
<?xml version="1.0" encoding="utf-8"?>
<produtos>
  <produto>
    <cod>LACT02330993</cod>
    <descrición>Leite enteira envase 1L</descrición>
  </produto>
  <produto>
    <cod>LACT00493112</cod>
    <descrición>Margarina vexetal tarrina 250g</descrición>
  </produto>
  …
</produtos>
```

And more ahead, the same company creates a new grammar for the regular clients, for example:

```xml
<?xml version="1.0" encoding="utf-8"?>
<clientes>
  <cliente>
    <cod>CL09384</cod>
    <nome>Uxío Fuentes Neira</nome>
    <enderezo>Rúa Europa 24, 3ºA</enderezo>
    <teléfono>555098433</teléfono>
  </cliente>
  <cliente>
    …
  </cliente>
  …
</clientes>
```

In both documents, there is an element called "cod". While customer data is not mixed with the product data, there will be no problem. But if we needed to create sales documents that contain the customer data and the data of the products they bought, we would have to differentiate the "cod" elements in some way according to their meaning.

The easiest way to do this would be to change the name of one (or both) of these elements, for example, "cod_produto" and "cod_cliente". But in many cases this solution is not valid; for example, it would be necessary to modify the applications that are working with the original name of the elements.

Namespaces are a W3C (World Wide Web Consortium) recommendation so common names of elements and tags do not collide, either because they come from different XML documents to a possible destination document, either because we are interested in separating in a structured way the contents of a document. In the previous example, we could use two namespaces: one for the products and one for the clients; in this way, we only have to make sure that each name is unique within its namespace. When we combine both "cod" elements, each one of them will be associated with their respective namespace and they will not collide.

We must use namespaces in our XML documents if there is a possibility that they would be shared and mixed with other XML documents.

Declaration of a namespace

To declare a namespace, the reserved word "`xmlns`" (**XML NameSpace**) is used. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<produtos xmlns="http://www.atendadepaco.com/espazosdenomes/produtos/">
  <produto>
    <cod>LACT02330993</cod>
    <descrición>Leite enteira envase 1L</descrición>
  </produto>
  <produto>
    <cod>LACT00493112</cod>
    <descrición>Margarina vexetal tarrina 250g</descrición>
  </produto>
  …
</produtos>
```

When we declare a namespace in the root element in this way, all elements of the document become part of the namespace. This is what is called a **default namespace**. The names of the elements in the document are composed of two parts: the namespace + the local name. Local names by themselves no longer identify the elements of the document.

It is important to note that declarations of namespaces use the same syntax as the attributes but are not considered attributes.

URIs, URLs and URNs

To identify a namespace and differentiate it from others, it is associated with a text string. This string must be unique, that is, it cannot be used to identify other namespaces.

One solution is to use a **URL** (Uniform Resource Locator) for the namespace's text string. An URL specifies the location of a resource, for example, a file on an FTP server or a web page:

http://www.atendadepaco.com/espazosdenomes/produtos/

In these cases, the resource referenced by the URL usually contains information about the namespace.

An URL is a special form of an **URI** (Uniform Resource Identifier). An URI is just an identifier; it does not have to be the address of an existing resource. It can have the format of an URL or any other.

The most common way to identify a namespace is by using a URI in the form of a URL. For example, if the resource identified by the previous URL did not exist, the address would continue to be valid as the identifier (URI) of the namespace:

http://www.atendadepaco.com/espazosdenomes/produtos/

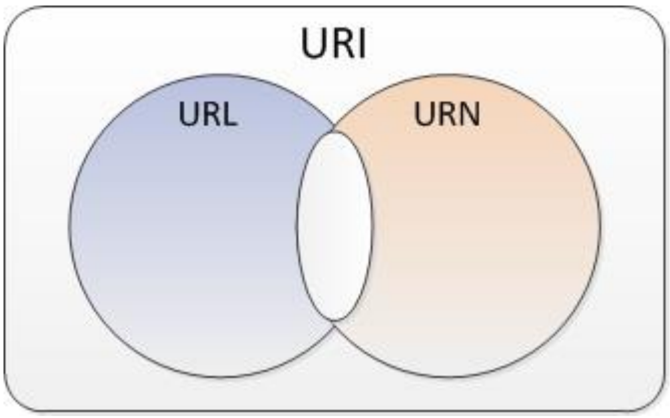
Another type of URI is an **URN** (Uniform Resource Name). An URN is a name that identifies something. Each URN consists of a namespace and a string that identifies an item in that namespace. Its form is:

`urn:[espazo de nomes]:[cadea de texto]`

The identifiers of the namespaces for the URN can be registered at the IANA (Internet Assigned Numbers Authority). For example, in the book ISBN namespace (isbn), an URN could be:

`urn:isbn:9780470114872`

In summary, the string that identifies a namespace must be a URI. Two special types of URI are URLs and URNs, although the latter are little used.



Namespaces with prefix

As we have seen, when in the root element of a document we use a default namespace, this affects the rest of the elements of the document. This is useful in the case of documents that contain a single namespace, such as the product or client documents we work with before.

When we need to use more than one namespace in an XML document, we must choose a prefix for them. The prefix is indicated next to the word "`xmlns`", separated by the "`:`" character, and it will be what we will have to use with those elements of the document to reflect that they belong to the given namespace.

For example, if we create a sales document, we could do it as follows:

```xml
<?xml version="1.0" encoding="utf-8"?>
<venda xmlns="http://www.atendadepaco.com/espazosdenomes/vendas/"
xmlns:cl="http://www.atendadepaco.com/espazosdenomes/clientes/"
xmlns:pr="http://www.atendadepaco.com/espazosdenomes/produtos/">
  <cl:cliente>
    <cl:cod>CL09384</cl:cod>
    <cl:nome>Uxío Fuentes Neira</cl:nome>
    <cl:enderezo>Rúa Europa 24, 3ºA</cl:enderezo>
    <cl:teléfono>555098433</cl:teléfono>
  </cl:cliente>
  <pr:produtos>
    <pr:produto>
      <pr:cod>LACT02330993</pr:cod>
      <pr:descrición>Leite enteira envase 1L</pr:descrición>
    </pr:produto>
    <pr:produto>
      <pr:cod>LACT00493112</pr:cod>
      <pr:descrición>Margarina vexetal tarrina 250g</pr:descrición>
    </pr:produto>
    …
  </pr:produtos>
  …
  <importe_total>16,34€</importe_total>
</venda>
```

Then:

- The default namespace for the document is that corresponding to the sales documents ("`http://www.atendadepaco.com/espazosdenomes/vendas/`").
- In the root element, other namespaces are also declared, indicating a prefix for them, and they are used by indicating that prefix in the elements of the document.
- When a prefix is used on an item, it must be made both on the opening label and on the closing label.
- We must indicate the prefix in all items belonging to the namespace. For example, the namespace "`http://www.atendadepaco.com/espazosdenomes/produtos/`" must be applied to the element "`produtos`", and to the subelements "`produto`", "`descrición`" and "`cod`" as well.
- The names of the prefixes must meet the same rules as the names of the elements and attributes, and besides, they cannot use the "`:`" character.

Scope of a namespace

The scope of a namespace is the set of elements of the document in which we can use it. When we declare a namespace in the root element, its scope is the entire document; we can use that namespace on any item in the document, including the root element.

But although it is a common practice to declare all the namespaces of a document in its root element, this is not necessary and, in many cases, it is also not easy. Within an XML document, you can declare namespaces (by default or by prefix) in any of its elements. The scope of a namespace covers the item in which it is declared, and all the elements that it contains.
It is imperative to declare a namespace in the same item in which we are going to use it, or in a parent element. For example, we could have made the previous sales document as follows:

```xml
<?xml version="1.0" encoding="utf-8"?>
<venda xmlns="http://www.atendadepaco.com/espazosdenomes/vendas/">
  <cl:cliente xmlns:cl="http://www.atendadepaco.com/espazosdenomes/clientes/">
    <cl:cod>CL09384</cl:cod>
    <cl:nome>Uxío Fuentes Neira</cl:nome>
    <cl:enderezo>Rúa Europa 24, 3ºA</cl:enderezo>
    <cl:teléfono>555098433</cl:teléfono>
  </cl:cliente>
  <pr:produtos xmlns:pr="http://www.atendadepaco.com/espazosdenomes/produtos/">
    <pr:produto>
      <pr:cod>LACT02330993</pr:cod>
      <pr:descrición>Leite enteira envase 1L</pr:descrición>
    </pr:produto>
    <pr:produto>
      <pr:cod>LACT00493112</pr:cod>
      <pr:descrición>Margarina vexetal tarrina 250g</pr:descrición>
    </pr:produto>
    …
  </pr:produtos>
  …
  <importe_total>16,34€</importe_total>
</venda>
```

We can redefine the default namespace, or a namespace associated with a prefix, within its scope. For this, we simply have to define it again. For example:

```xml
<?xml version="1.0" encoding="utf-8"?>
<venda xmlns="http://www.atendadepaco.com/espazosdenomes/vendas/">
  <cliente xmlns="http://www.atendadepaco.com/espazosdenomes/clientes/">
    <cod>CL09384</cod>
    <nome>Uxío Fuentes Neira</nome>
    <enderezo>Rúa Europa 24, 3ºA</enderezo>
    <teléfono>555098433</teléfono>
  </cliente>
  <produtos xmlns="http://www.atendadepaco.com/espazosdenomes/produtos/">
    <produto>
      <cod>LACT02330993</cod>
      <descrición>Leite enteira envase 1L</descrición>
    </produto>
    <produto>
      <cod>LACT00493112</cod>
      <descrición>Margarina vexetal tarrina 250g</descrición>
    </produto>
    …
  </produtos>
  …
  <importe_total>16,34€</importe_total>
</venda>
```

We can also delete the previous definition of a default namespace, or also (from XML version 1.1) of a prefixed namespace. We only must redefine it by using an empty string as the identifier.

```
<?xml version="1.0" encoding="utf-8"?>
<elemento_raiz xmlns="http://www.exemplo.com/">
  <elemento1 xmlns="">
    …
  </elemento1>
  …
</elemento_raiz>
```

In the previous example, the declared default namespace in the root element does not apply to the element "`<elemento1>`" nor to any of its subelements.

Attributes and namespaces

Namespaces as we have seen apply only to the elements. They do not apply to attributes; and the other elements of an XML document, such as comments or processing instructions, do not have namespaces. For example, in the following document:

```
<?xml version="1.0" encoding="utf-8"?>
<elemento_raiz xmlns:ns="http://www.exemplo.com/">
  <ns:elemento1 cod="A334">
    …
  </ns:elemento1>
  …
</elemento_raiz>
```

The namespace "`http://www.exemplo.com/`" is declared in the root element and is applied to the element "`<elemento1>`"; this does not include the attribute "`cod`". In XML documents, the attributes belong to a particular element, and this prevents them from colliding with attributes belonging to other namespaces.

In any case, if we want to apply a namespace to an attribute, we could use the same syntax as for elements:

```
<?xml version="1.0" encoding="utf-8"?>
<elemento_raiz xmlns:ns="http://www.exemplo.com/">
  <ns:elemento1 ns:cod="A334">
    …
  </ns:elemento1>
  …
</elemento_raiz>
```