

**DTD** Document type Definition es un lenguaje descriptor y de definición de gramáticas para los atributos de los documentos XML, que describe la forma válida y estructurada que deben tener para por ejemplo ser consumido por un sistema. Esto garantiza la interoperabilidad y preve errores en el sistema.

<!DOCTYPE root-element [...] > nombre del elemento raíz/principal del XML

<!ELEMENT element-name category quantificadores > Cada etiqueta/elemento se

} EMPTY  
#PCDATA  
ANY

Indican cuántas veces aparece un hijo  
" " = 1 | "+" = 1 v más | "\*" = Ø v más | "?" = Ø v 1

<!ATTLIST element-name attribute-name attribute-type attribute-value >

CDATA	#IMPLIED
ENUM	#REQUIRED
IDREF	
ID	

## Ejemplo:

```
<?xml version="1.0"?>
<!DOCTYPE bookstore [
    <!ELEMENT bookstore ( book | magazine)*>
    <!ELEMENT book ( title, authors, remark?)>
    <!ATTLIST book ISBN CDATA #REQUIRED
                    Price CDATA #REQUIRED
                    Edition CDATA #IMPLIED>
    <!ELEMENT magazine (title)>
    <!ATTLIST magazine Month CDATA #REQUIRED
                    Year CDATA #REQUIRED>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT remark (#PCDATA)>
    <!ELEMENT authors (author+)>
    <!ELEMENT author (firstname, lastname)>
    <!ELEMENT firstname (#PCDATA)>
    <!ELEMENT lastname (#PCDATA)>
]>
```

```
<bookstore>
    <book ISBN = "1234xyz" Price = "10"
          Edition = "3rd">
        <title>1984 </title>
        <remark>
            A dystopian and political story
        </remark>
        <authors>
            <author>
                <firstname>George</firstname>
                <lastname>Orwell</lastname>
            </author>
        </authors>
    </book>
    <magazine month = "may" year = "2025">
        <title>Vogue </title>
    </magazine>
</bookstore>
```

# Ejercicio 1:

```
<!DOCTYPE TVSCHEDULE [  
<!ELEMENT TVSCHEDULE (CHANNEL+)>  
<!ELEMENT CHANNEL (BANNER, DAY+)>  
<!ELEMENT BANNER (#PCDATA)>  
<!ELEMENT DAY (DATE, (HOLIDAY|PROGRAMSLOT+)+)>  
<!ELEMENT HOLIDAY (#PCDATA)>  
<!ELEMENT DATE (#PCDATA)>  
<!ELEMENT PROGRAMSLOT (TIME, TITLE, DESCRIPTION?)>  
<!ELEMENT TIME (#PCDATA)>  
<!ELEMENT TITLE (#PCDATA)>  
<!ELEMENT DESCRIPTION (#PCDATA)>  
  
<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>  
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>  
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>  
<!ATTLIST TITLE RATING CDATA #IMPLIED>  
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>  
]>
```

```
<TVSCHEDULE name = "mornings" >  
<CHANNEL chan = "312" >  
<BANNER> disney channel </BANNER>  
<DAY>  
<DATE>04/05/2025 </DATE>  
<PROGRAMSLOT>  
<TIME> 10:00 </TIME>  
<TITLE> Hannah Montana </TITLE>  
</PROGRAMSLOT>  
</DAY>  
</CHANNEL>  
</TVSCHEDULE>
```

## Ejercicio 2:

Crea un DTD que valide el siguiente XML:

```
<articles>
  <article id="x34675">
    <name>Apache Spark Architecture</name>
    <month>december</month>
    <author name="kay vennisla"/>
    <reviews lang=""/>
    <feedback> high rating</feedback>
    <reviews lang="de">The best content with diagrams</reviews>
  </article>
</articles>
```

```
<?xml version="1.0"?>
<!DOCTYPE articles [
  <!ELEMENT articles ( article+ )>
  <!ELEMENT article ( name, month,
                      author, reviews*, feedback )>
  <!ATTLIST article id CDATA #REQUIRED>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT month (#PCDATA)>
  <!ELEMENT author EMPTY>
  <!ATTLIST author name CDATA #REQUIRED>
  <!ELEMENT reviews (#PCDATA)>
  <!ATTLIST reviews lang CDATA #REQUIRED>
  <!ELEMENT feedback (#PCDATA)>
]>
```

# Ejemplo con Punteros

## Archivo dtd\_ejemplo4.xml:

```
<?xml version="1.0"?>
<!DOCTYPE Bookstore [
    <!ELEMENT Bookstore (Book*, Author*)>
    <!ELEMENT Book (Title, Remark?)>
    <!ATTLIST Book ISBN ID #REQUIRED
            Price CDATA #REQUIRED
            Authors IDREFS #REQUIRED>
    <!ELEMENT Title (#PCDATA)>
    <!ELEMENT Remark (#PCDATA | BookRef)*>
    <!ELEMENT BookRef EMPTY>
    <!ATTLIST BookRef book IDREF #REQUIRED>
    <!ELEMENT Author (First_Name, Last_Name)>
    <!ATTLIST Author Ident ID #REQUIRED>
    <!ELEMENT First_Name (#PCDATA)>
    <!ELEMENT Last_Name (#PCDATA)>
]>
```

```
<!DOCTYPE boosture SYSTEM "dtd-ejemplo4.xml">
< bookstore>
    < book ISBN="3312" Price="30" Authors="JMB">
        < title> Peter Pan </title>
        < remark>
            < bookref book="3312" />
        </remark>
    </book>
    < author ident="JMB">
        < firstname> James M. </firstname>
        < lastname> Barrie </lastname>
    </author>
</bookstore>
```

```

<university name="UPM">
  <course id="CS101" title="Algoritmos Básicos">
    <credits>6</credits>
    <prerequisites/>
  </course>
  <course id="CS102" title="Estructuras de Datos">
    <credits>6</credits>
    <prerequisites>CS101</prerequisites>
  </course>
  <course id="CS201" title="Optimización">
    <credits>6</credits>
    <prerequisites>CS102 CS101</prerequisites>
  </course>
  <student id="s100" name="María López" enrolled="CS101 CS102"/>
  <student id="s101" name="Juan Pérez" enrolled="CS201"/>
</university>

```

!DOCTYPE university [
 !ELEMENT university (course+, student+)
 !ATTLIST university name CDATA #REQUIRED
 !ELEMENT course ( credits , prerequisites )
 !ATTLIST course id ID #REQUIRED
 title CDATA #REQUIRED
 !ELEMENT credits #PCDATA
 !ELEMENT prerequisites (#PCDATA)\*
 !ELEMENT student EMPTY
 !ATTLIST student id CDATA #REQUIRED
 name CDATA #REQUIRED
 enrolled CourseRef #IMPLIED
 IDREFS
 ]>

## dtdfiles.dtd

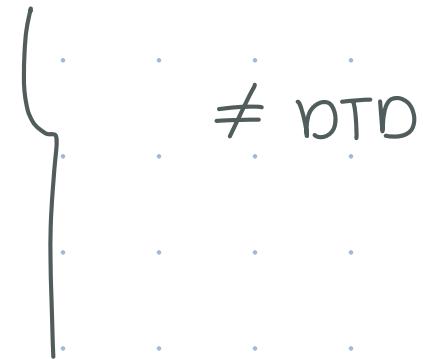
```
<!ENTITY % perms "(read|write|execute)">
<!ELEMENT filesystem (file+)>
<!ELEMENT file EMPTY>
<!ATTLIST file
  name    CDATA      #REQUIRED
  size    CDATA      #REQUIRED
  owner   CDATA      #REQUIRED
  perms   %perms;   #REQUIRED
  tags    NMTOKENS   #IMPLIED
>
```

```
<?xml version="1.0"?>
<!DOCTYPE Filesystem SYSTEM "dtdfiles.dtd">
<Filesystem>
  <file name="racetrack" size="2048"
        owner="Urianna" perms="write"
        tags="thesis" />
  <file name="racetrack" size="2048"
        owner="Guillermo" perms="read"
        tags="thesis" />
</Filesystem>
```

## XSD o XML Schema Definition

teóricamente hablando cumple la misma función descriptiva y de definición para XML, sin embargo, este lenguaje se podría decir que es más especializado.

Escrito en XML, permite detallar tipos cantidades específicas, duplicar definiciones, es extensible y todo esto permite un mayor control sobre el XML



un nodo es cualquier cosa dentro de un XML estq.,  
atrib., txt...j

# XPATH

es un lenguaje de consultas para navegar por los nodos  
de un XML tratándolo como un **árbol (DOM)**

Se construyen como directorios:

\$ / → ref. al nodo raíz

\$ // → "yo" y cualquier elemento descendiente

\$ /@X → sea X un atributo

\$ ../ → ir hacia atrás

...

## Ejercicios:

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
  <name>L.1.1.1</name>
  <student id="035">
    <name>Mark</name>
    <year>1999</year>
  </student>
  <student id="007">
    <name>Ana</name>
    <year>1998</year>
  </student>
</class>
```

- 1 • ¿La información de toda la clase?
- 2 • ¿Todos los estudiantes?
- 3 • ¿Todos los nombres de los estudiantes?
- 4 • ¿Todas las ids de los estudiantes?

1. \$ /class
2. \$ /class/student
3. \$ /class/student/name
4. \$ /class/student/@id

\$ student → C nodo "student"  
\$ //name → C elemento name  
\$ //@id → C atributo id

C = cualquier

Los Predicados [ ] son formas de anidar condiciones para filtrar:

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
  <name>L.1.1.1</name>
  <student id="035">
    <name>Mark</name>
    <year>1999</year>
  </student>
  <student id="007">
    <name>Ana</name>
    <year>1998</year>
  </student>
</class>
```

1. \$ /class/student[2]
2. \$ /class/student[@id = "035"]
3. \$ /class/student[year ≥ 1999]
4. \$ /class/student[@\*]

Las funciones tmb. se pueden combinar con predicados:

name() text() last() contains() count()

Ejemplos de navegación:

self parent descendant

preceding-sibling following-sibling

1. ¿El segundo estudiante?
2. ¿Estudiante con id 035?
3. ¿Estudiantes del año 1999 o posteriores?
4. ¿Todos los elementos student que tengan al menos un atributo?

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
<name>L.1.1.1</name>
<student id="035">
    <name>Mark</name>
    <year>1999</year>
</student>
<student id="007">
    <name>Ana</name>
    <year>1998</year>
</student>
</class>
```

1. \$ count(//students)
2. \$ //student[year="1999"] /name/text()
3. \$ //student[contains(text(),"Ana")]
 .. /year/text()
4. \$ //student[last()]

- 1 • ¿Cuántos estudiantes hay?
- 2 • ¿Nombre de los estudiantes que han nacido en 1999?
- 3 • ¿Año de nacimiento de Ana?
- 4 • ¿Último estudiante?

# Ejercicio 1 – Biblioteca

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<biblioteca>
    <libro id="b1">
        <titulo>Programación en Java</titulo>
        <autor>Laura García</autor>
        <precio>25.50</precio>
    </libro>
    <libro id="b2">
        <titulo>Introducción a XML</titulo>
        <autor>Eduardo Ruiz</autor>
        <precio>18.00</precio>
    </libro>
    <libro id="b3">
        <titulo>Avanzado en XPath</titulo>
        <autor>Laura García</autor>
        <precio>30.00</precio>
    </libro>
</biblioteca>
```

1. Todos los títulos de libro.
2. El precio del libro con `@id="b2"`.
3. Libros cuyo precio sea mayor que 20.
4. ID y título de los libros escritos por "Laura García".
5. Número total de libros.



1. `$ /biblioteca/libro/titulo`
1. `$ //libro/titulo`
2. `$ /biblioteca/libro[@id="b2"]/precio/text()`
3. `$ //libro[ precio >= 20 ]`
4. `$((//libro [ autor = "Laura García" ]/titulo) | ((//libro [ autor = "Laura García" ]/@id))`
5. `$ count (//libro)`

## Ejercicio 2 – Catálogo de música

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
  <album año="1995">
    <titulo>Thriller</titulo>
    <artista>Michael Jackson</artista>
    <cancion>Beat It</cancion>
    <cancion>Billie Jean</cancion>
  </album>
  <album año="2000">
    <titulo>Hybrid Theory</titulo>
    <artista>Linkin Park</artista>
    <cancion>In the End</cancion>
    <cancion>Numb</cancion>
    <cancion>One Step Closer</cancion>
  </album>
</catalogo>
```

1. Todos los álbumes del año 2000.
2. Cantidad de canciones en el álbum "Thriller".
3. Títulos de todos los álbumes con más de 2 canciones.
4. Artista del primer álbum.
5. Álbumes lanzados antes de 1998.

1. \$ //album[@año = 2000]
2. \$ count( //album[título = "Thriller"] ) / canción
3. \$ //album[ count(cancion) > 2 ] / título
4. \$ //album[1] / artista
5. \$ //album[number(@año) < 1998]

## Ejercicio 3 – Órdenes de compra

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ordenes>
  <orden num="1001">
    <cliente>Carlos</cliente>
    <item cantidad="2">
      <producto>Teclado</producto>
      <precio>20</precio>
    </item>
    <item cantidad="1">
      <producto>Monitor</producto>
      <precio>150</precio>
    </item>
  </orden>
  <orden num="1002">
    <cliente>Ana</cliente>
    <item cantidad="5">
      <producto>USB</producto>
      <precio>5</precio>
    </item>
  </orden>
</ordenes>
```

1. Número de órdenes en total.
2. Todos los números de orden ( @num ).
3. Nombre de cliente de la orden número 1001.
4. Todos los productos comprados por Ana.
5. Órdenes que incluyen al menos un ítem con @cantidad > 3 .

1. \$ count ( //orden )

2. \$ //orden/@num

3. \$ //orden [number(@num) = 1001] /cliente /text()

4. \$ //orden [cliente = "Ana"] /item /producto

5. \$ //orden /item [number(@cantidad) > 3 ]

## Ejercicio 5 – XML con espacios de nombres

xml

```
<?xml version="1.0"?>  
<root xmlns:p="http://ejemplo.org/personas"  
      xmlns:a="http://ejemplo.org/animales">  
  <p:persona>  
    <p:nombre>Luís</p:nombre>  
    <p:edad>28</p:edad>  
  </p:persona>  
  <a:animal>  
    <a:especie>Perro</a:especie>  
    <a:nombre>Firulais</a:nombre>  
  </a:animal>  
</root>
```

1. Seleccionar todos los nodos `<p:persona>`.
2. El nombre de la persona (`//p:persona/p:nombre`).
3. Todos los `<a:animal>` y su especie.
4. Edad de la persona solo si es mayor de 25.
5. Contar cuántos elementos están en el namespace de "animales".

1. \$ /root/p:persona

2. \$ /root/p:persona/p:nombre/text()

3. \$(/root/a:animal/) |(/root/a:animal/  
a:especie/text())

4. \$ /root/p:persona[p:edad > 25]/p:edad/text()

5. \$ count(/root/a:animal)

5! \$ count(/root/a:\*)