

XML son las siglas de *Extensible Markup Language* y desde hace unos años se ha convertido en un estándar para el manejo de datos debido a la facilidad con la que los propios usuarios pueden crear sus etiquetas, conceptualmente similares a las de HTML, pero que sirven para organizar la información de forma semántica a la vez que estructurada. Los diferentes sistemas que pueden acceder a dicha estructura no tienen porque saber nada previamente de la misma, pero si ha sido creada de forma correcta, podrá acceder y modificar los datos sin tener que modificar los ya existentes.

Desde la versión 5 de Flash Player, ActionScript ha puesto a disposición de los desarrolladores herramientas para manejar, crear y alterar estructuras de datos basadas en XML. En ActionScript 1 y ActionScript 2 se trabajaba con estructuras XML en base a las prestaciones ofrecidas por la clase XML. Dicha clase estaba basada en el W3C Document Object Model (DOM), un estándar del W3C para interactuar con documentos XML.

En ActionScript 3 la forma de trabajar con documentos XML ha cambiado completamente siendo mucho más sencillo, intuitivo y eficiente. ActionScript 3 implementa E4X (ECMAScript for XML), una extensión oficial para el estándar ECMA-262 que incluye el XML como un tipo de datos nativo en el lenguaje.

Aunque en ActionScript 3 se puede seguir utilizando el sistema basado en DOM, trabajar con E4X aporta muchas ventajas y comodidades al desarrollador.

**Nota:** El cambio de AS2 a AS3 puede ser un poco confuso en lo que refiere al nombre de las clases XML. En AS2 la clase XML permitía tratar con documentos XML utilizando DOM.

*En AS3 la clase XML accede a los documentos utilizando E4X. Si en AS3 quiere utilizar DOM deberá utilizar la clase XMLDocument*

## 12.1. Definición de un fichero XML

Antes de empezar a manejar y manipular documentos XML, va a ver qué aspecto y estructura tiene un fichero XML.

Un fichero XML consiste en un fichero de texto que puede crear en cualquier editor de texto simple como el bloc de notas, NotePad++, ultraEdit, jEdit, etc.

```
<articulos>
<boligrafos>
<modelo referencia="1" cantidad="10" precio="3300" />
<modelo referencia="2" cantidad="7" precio="2500" />
<modelo referencia="3" cantidad="9" precio="6000" />
</boligrafos>

<plumas>
<modelo referencia="4" cantidad="20" precio="5500" />
<modelo referencia="5" cantidad="25" precio="7500" />
<modelo referencia="6" cantidad="15" precio="10000" />
<modelo referencia="7" cantidad="36" precio="20000"/>
</plumas>
</articulos>
```

Si tenemos una papelería que vende plumas y bolígrafos, acabamos de estructurar la información en forma de árbol, de manera ordenada y jerárquica por cuanto para acceder a la referencia de un determinado modelo antes hemos de pasar por si es bolígrafo o pluma. <articulos>, <boligrafos>, <plumas> son lo que se denominan nodos del árbol XML o etiquetas XML. Por cada uno de ellos que se abre, luego existe otro que se cierra, por ejemplo, </articulos>.

Los nodos <modelo ... /> también son nodos pero a diferencia que los otros nodos, los nodos <modelo .../> no contienen a otros subnodos ni existe una etiqueta </modelo> que indique el cierre del nodo. En XML existe la posibilidad de que si un nodo no contiene subnodos, el nodo se puede cerrar en la misma etiqueta que lo abre. La / al final de los nodos <modelo .../> indica el cierre del nodo. Hubiera sido igualmente válido cerrar los nodos de la siguiente forma:

```
<modelo referencia="7" cantidad="36"precio="20000">
</modelo>
```

Otra peculiaridad de los nodos `<modelo .../>` es que poseen una serie de cualidades como son referencia, cantidad y precio. Esas cualidades se denominan atributos de un nodo. Los atributos son propiedades del nodo en el que se definen.

Los nodos de un XML siguen un anidamiento de tal forma que un nodo puede ser el padre de otros a la vez que puede ser el hijo de otros. En el ejemplo diríamos que el nodo `<boligrafos>...</boligrafos>` es un hijo de `<articulos>` y es el padre de varios nodos `<modelo .../>`. En un documento XML bien estructurado todos los nodos tienen que tener un padre, exceptuando el primer nodo (en el ejemplo `<articulos/>`).

El documento XML que hemos definido sigue una estructura determinada, en este caso definida por nosotros. Pero según las necesidades de cada aplicación y desarrollador la estructura de XML puede cambiar. Por ejemplo, podríamos haber definido la estructura de tal forma que en vez de utilizar atributos para la referencia, cantidad y precio hubiéramos utilizado nodos:

```
<modelo>
<referencia>4</referencia>
<cantidad>20</cantidad>
<precio>5500</precio>
</modelo>
```

## Definiendo un objeto XML

Tal como hemos apuntado, una de las grandes mejoras introducidas por ActionScript 3 en lo que refiere al tratamiento de XML recae en que se ha convertido en un tipo de datos nativo del lenguaje.

De esta forma existen tres formas diferentes de declarar un objeto XML:

1. Declaración explícita en el código.
2. Declaración a partir de un String.
3. Utilizar los métodos de manipulación de la clase XML.

## 12.2. Declaración explícita

De la misma forma que declaramos un variable de tipo `String` o de tipo `Number`, podemos definir una variable de tipo XML y asignarle una estructura de datos basada en XML.

Para ver el ejemplo, abra el documento `cap12/ej1/declaracionInline fla` del material descargado. En este ejemplo

se ha definido un nodo padre `<countries>` que engloba una serie de países marcados con el nodo `<country/>`.

```
var paises:XML = <countries>
<country name="Afghanistan"/>
<country name="Albania"/>
<country name="Algeria"/>
<country name="American_samoa"/>
<country name="Andorra"/>
<country name="Angola"/>;
</countries>;
trace (paises);
```

Cuando ejecute el código anterior verá en la pantalla de salida el XML que se ha declarado en la variable `paises`.

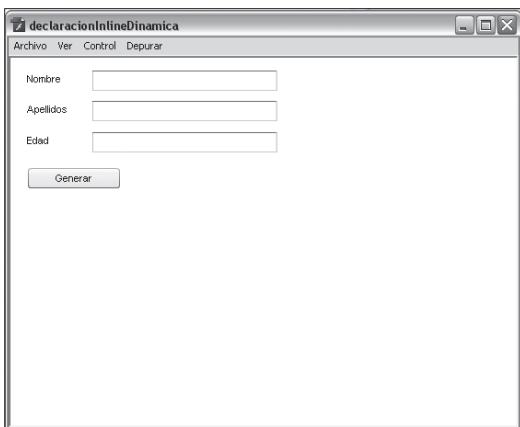
Los casos en los que generará un XML estático directamente en el código serán pocos. Cuando tenga que cargar datos XML normalmente los cargará de ficheros externos para disminuir así el tamaño de la película que genere pero también para poder generar de forma dinámica esos datos desde el servidor.

La declaración de datos XML en el código se suele utilizar para enviar datos estructurados al servidor. En estos casos los datos no se conocen en tiempo de desarrollo sino que dependen en muchas ocasiones de datos que el usuario introduzca por ejemplo en un formulario.

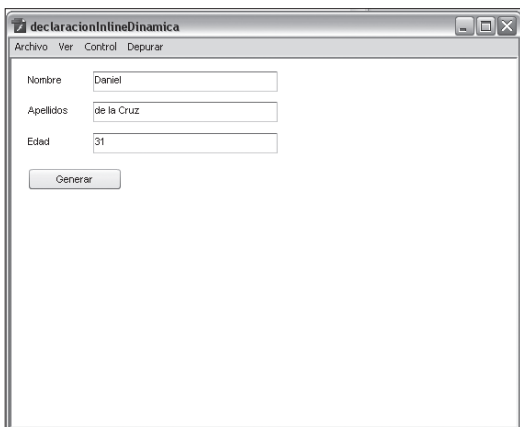
En el documento `cap12/ej2/declaracionInlineDinamica fla` del material descargado encontrará el código de un ejemplo en el que se le pide al usuario, a través de un formulario, su nombre, apellidos y edad. Al pulsar el botón **generar**, estos datos se almacenan en una estructura XML y se pintan en la pantalla de salida.

En el escenario de la aplicación existen 3 componentes `TextInput` con nombres de instancia `nombreFld`, `apellidosFld` y `edadFld` que sirven al usuario para introducir su nombre, apellidos y edad respectivamente. A su vez existe un botón con nombre de instancia `generarBtn`.

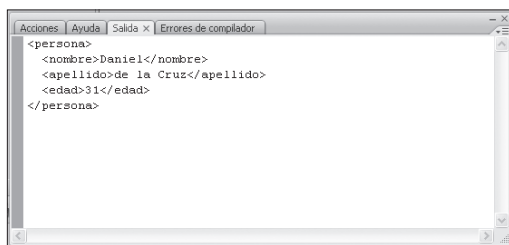
```
generarBtn.addEventListener (MouseEvent.CLICK,
generarXML);
function generarXML (event:MouseEvent):void
{
var persona:XML = <persona id={identificador.text}>
<nombre>{nombreFld.text}</nombre>
<apellido>{apellidosFld.text}</apellido>
<edad>{edadFld.text}</edad>
</persona>;
trace (persona);
}
```



**Figura 12.1.** Formulario de petición de datos.



**Figura 12.2.** Introducción de datos en el formulario.



**Figura 12.3.** Datos generados en la Ventana de Salida.

En el escenario de la película flash verá que se han arrastrado tres instancias de un campo de texto y se les ha asignado el nombre de instancia `nombreFld`, `apellidosFld` y `edadFld`. Además se ha añadido un botón con nombre de instancia `generarBtn`.

En el código se ha definido un listener de tal forma que cuando el usuario haga clic en el botón **generar** se ejecute la función `generarXML()`.

Si se fija ahora en la forma en la que se ha definido el XML verá que se han definido los valores de los nodos `nombre`, `apellido` y `edad` utilizando expresiones entre llaves (`{ y }`). Las llaves indican a ActionScript que lo que tienen en medio se trata de una expresión que se tiene que evaluar. En el caso del ejemplo las expresiones hacen referencia a los valores de los distintos campos de texto que se han añadido en la aplicación.

Utilizando este sistema se pueden dinamizar tanto los valores y nombre de los nodos como los nombres de los atributos y sus valores de tal forma que podríamos hacer cosas del estilo del ejemplo que encontrará en el documento `cap12/ej3/declaracionInlineMuyDinamica fla`:

```
var nodoPersona:String = "PERSONA";
var nodoNombre:String = "NOMBRE";
var nodoApellidos:String = "APELLIDOS";
var nodoEdad:String = "EDAD";
var atributoId:String = "id";
var persona:XML = <{nodoPersona} {atributoId}={idFld.
text}> <{nodoNombre}>{nombreFld.text}</{nodoNombre}>
<{nodoApellidos}>{apellidosFld.text}</{nodoApellidos}>
<{nodoEdad}>{edadFld.text}</{nodoEdad}>
</{nodoPersona}>;
trace (persona);
```

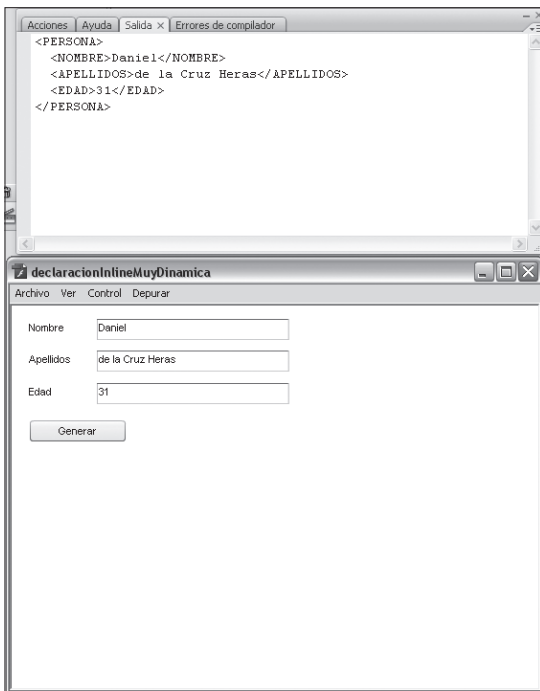
A continuación vamos a extender el funcionamiento de este pequeño formulario para que cada vez que pulsemos el botón se añada el XML descriptivo de una persona a un listado XML de personas. Para ver el ejemplo, abra el documento `cap12/ej4/declaracionInlineListadoPersonas fla` del material descargado.

```
anadirBtn.addEventListener (MouseEvent.CLICK,
generarXML);
verBtn.addEventListener (MouseEvent.CLICK, verXML);
function generarXML (event:MouseEvent):void
{
var persona:XML = <persona>
<nombre>{nombreFld.text}</nombre>
<apellidos>{apellidosFld.text}</apellidos>
```

```

<edad>{edadFld.text}</edad>
</persona>;
personas.appendChild (persona);
nombreFld.text = "";
apellidosFld.text = "";
edadFld.text = "";
}
function verXML (event:MouseEvent):void
{
trace (personas);
}
var personas:XML = <personas></personas>;

```



**Figura 12.4.** Datos introducidos en el formulario y generados en la Ventana de Salida.

La novedad que hemos introducido en el código es el uso del método `appendChild`. Este método permite añadir como nodo hijo un nodo XML a un nodo padre. En el caso del ejemplo estamos añadiendo el XML `<persona>...</persona>` al nodo padre `<personas/>`.

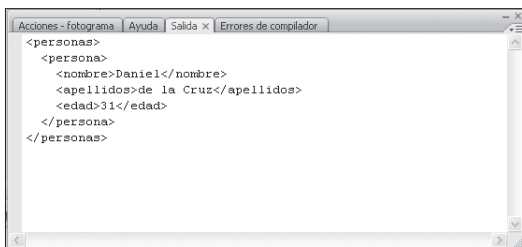


Figura 12.5. Datos generados en la Ventana de Salida.


## 12.3. Declaración a partir de un String

En un gran número de casos, los datos XML que cargue en su aplicación serán datos dinámicos y cambiantes entre invocaciones. En muchas ocasiones serán datos generados dinámicamente por páginas php, java, python, ruby o cualquier otro lenguaje de servidor. En estos casos no le será posible declarar los datos directamente en el código como en el ejemplo anterior.

En estos casos deberá cargar los datos de forma dinámica utilizando para ello la clase `XMLLoader` de la forma que se detalla en el siguiente ejemplo. Para ver el ejemplo, abra el documento `cap12/ej5/declaracionExterna.fla` del material descargado.

```
var articulosRequest:URLRequest = new URLRequest
("articulos.xml");
var xmlLoader:XMLLoader = new XMLHttpRequest ();
xmlLoader.addEventListener (Event.COMPLETE,
datosCargados);
xmlLoader.load (articulosRequest);
function datosCargados (event:Event):void
{
var articulosXML:XML = XML (event.target.data);
trace (articulosXML);
}
```

En la aplicación de ejemplo, a través del objeto `articulosRequest` definimos que queremos cargar el fichero externo `articulos.xml` y lo cargamos con el objeto `xmlLoader`. Recordando que las cargas de datos externos siempre son de carácter asincrónico tenemos que hacer uso de los eventos para que una vez cargados los datos se ejecute la función `datosCargados`.

A screenshot of a software window titled "Acciones - fotograma" with a menu bar containing "Ayuda", "Salida x", and "Errores de compilador". The main area displays XML code. The code is as follows:

```
<articulos>
  <boligrafos>
    <modelo referencia="1" cantidad="10" precio="3300"/>
    <modelo referencia="2" cantidad="7" precio="2500"/>
    <modelo referencia="3" cantidad="9" precio="6000"/>
  </boligrafos>
  <plumas>
    <modelo referencia="4" cantidad="20" precio="5500"/>
    <modelo referencia="5" cantidad="25" precio="7500"/>
    <modelo referencia="6" cantidad="15" precio="10000"/>
    <modelo referencia="7" cantidad="36" precio="20000"/>
  </plumas>
</articulos>
```

Figura 12.6. Datos generados en la Ventana de Salida.

Dentro del listener convertimos el String devuelto por la carga en una variable de tipo XML con la sintaxis XML (`string_a_convertir`).

## 12.4. Objetos XML y XMLList

Desde el punto de vista de ActionScript, podemos clasificar los objetos XML como simples o complejos. Un objeto XML será simple si contiene un atributo, un comentario, una *processing instruction* o un nodo de texto. El objeto será complejo si contiene cualquier combinación de elementos simples.

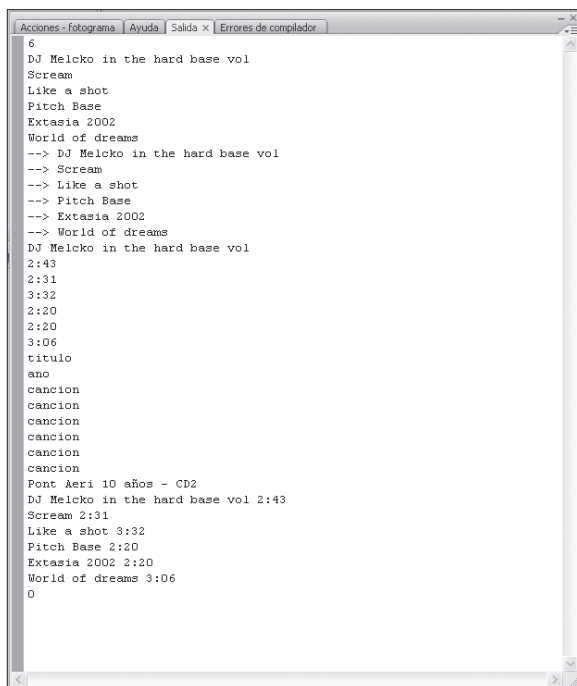
Una instancia de `XMLList` consiste en una colección de objetos XML de distinta naturaleza. Puede contener documentos XML completos, sólo fragmentos o resultados de una consulta XML. De una forma un tanto ruda podríamos decir que un `XMLList` es un Array en el que cada uno de los elementos es un objeto de tipo XML.

## 12.5. Recorriendo una estructura XML

Con el propósito de poder acceder a los distintos atributos y nodos de un XML, E4X incorpora el operador punto (`.`), el operador atributo (`@`) y el operador descendiente (`..`).

A partir de ahora los ejemplos se basarán en el XML que puede encontrar en el fichero `cap12/ej6/album.xml` o

en el fichero cap12/ej2/XML\_XMLList.fla del material descargado:

A screenshot of a software window titled 'Errores de compilador' (Compiler Errors). The window contains a list of XML data elements. The first part shows a list of tracks with their titles and durations: 'DJ Melcko in the hard base vol' (2:43), 'Scream' (2:31), 'Like a shot' (3:32), 'Pitch Base' (2:20), and 'Extasia 2002' (2:20). Below this, there are several 'cancion' (song) elements with their respective titles and durations. The window also shows a 'titulo' (title) element and an 'ano' (year) element. The window has a standard menu bar with 'Acciones - fotograma', 'Ayuda', 'Salida x', and 'Errores de compilador'.

```
6
DJ Melcko in the hard base vol
Scream
Like a shot
Pitch Base
Extasia 2002
World of dreams
--> DJ Melcko in the hard base vol
--> Scream
--> Like a shot
--> Pitch Base
--> Extasia 2002
--> World of dreams
DJ Melcko in the hard base vol
2:43
2:31
3:32
2:20
2:20
3:06
titulo
ano
cancion
cancion
cancion
cancion
cancion
cancion
Pont Aeri 10 años - CD2
DJ Melcko in the hard base vol 2:43
Scream 2:31
Like a shot 3:32
Pitch Base 2:20
Extasia 2002 2:20
World of dreams 3:06
0
```

**Figura 12.7.** Datos del documento mostrados en la Ventana de Salida.

```
var album:XML = <album>
<titulo>Pont Aeri 10 años - CD2</titulo>
<ano>2002</ano>
<cancion track="1" duracion="2:43">
<titulo>DJ Melcko in the hard base vol</titulo>
<artista>DJ Pildo</artista>
</cancion>
<cancion track="2" duracion="2:31">
<titulo>Scream</titulo>
<artista>DJ Sisu & Dj Artuño</artista>
</cancion>
<cancion track="3" duracion="3:32">
<titulo>Like a shot</titulo>
<artista>System 3</artista>
</cancion>
<cancion track="4" duracion="2:20">
```

```

<titulo>Pitch Base</titulo>
<artista>DJ Gere & Mak Attack</artista>
</cancion>
<cancion track="5" duracion="2:20">
<titulo>Extasia 2002</titulo>
<artista>Flashback</artista>
</cancion>
<cancion track="6" duracion="3:06">
<titulo>World of dreams</titulo>
<artista>Spark</artista>
</cancion>
</album>;

```

Este XML estructura la información de un disco de música conteniendo un listado de todas las canciones que contiene así como el nombre del disco y el año de publicación.

Si quisiéramos acceder a la lista de todas las canciones del disco sería equivalente a obtener un `XMLList` que contuviera los nodos XML `<cancion/>`.

```
var canciones:XMLList = album.cancion;
```

El operador punto (.) permite acceder a los nodos hijo que tengan como nombre de etiqueta el indicado. De esta forma `album.cancion` accede a todos los nodos `<cancion/>` dentro del XML de `album`.

El objeto `XMLList` `canciones` contendrá un total de 6 objetos XML. El método `length()` nos dará precisamente esa información

```
trace (canciones.length()); // 6
```

Para recorrer los elementos de un `XMLList` puede hacerlo utilizando un bucle de la misma forma que haría para recorrer un `Array`.

```
for (var i:int = 0; i < canciones.length (); i++)
{
trace (canciones [i].titulo);
}
```

El contenido de canciones `[i]` será un objeto XML `<cancion/>`. De esta forma, en la primera iteración del bucle, tomará el valor:

```

<cancion track="1" duracion="2:43">
<titulo>DJ Melcko in the hard base vol</titulo>
<artista>DJ Pildo</artista>
</cancion>

```

Al tratarse de un objeto XML podemos utilizar otra vez el operador punto (.) para acceder a su nodo hijo `<titulo/>`. En

la ventana de salida veremos todos los títulos de las canciones del álbum.

Otra forma de iterar los elementos del `XMLList` sería utilizando un bucle especializado `for each..in`. Este tipo de iteradores iteran las colecciones almacenando el valor para cada iteración en una variable en vez de almacenar el índice de la posición que ocupan en la posición.

```
for each (var cancion:XML in canciones)
{
trace ("-->", cancion.titulo);
}
```

Si quisiera acceder directamente a la primera canción del disco podría utilizar la siguiente expresión:

```
trace (album.cancion[0].titulo); // DJ Melcko in the
hard base vol
```

Hasta ahora sólo hemos accedido a los elementos de texto de nodos XML. El operador atributo (`@`) permite acceder a los atributos de un nodo.

La siguiente expresión permite acceder a un `XMLList` de los atributos `duracion` de los nodos `<cancion/>`. Cada elemento se corresponderá con un objeto XML que contenga el valor del atributo `duracion`. Tal como hemos visto antes podría iterar por el contenido del `XMLList` resultante utilizando cualquiera de los bucles presentados.

```
var duraciones:XMLList = album.cancion.@duracion;
```

Otra forma de acceder los nodos hijo de un documento XML es utilizando los métodos de la clase XML. El método `children ()` devuelve un `XMLList` que contiene todos los objetos XML hijos.

```
var hijos:XMLList = album.children ();
trace (hijos.length());
```

A diferencia del sistema que hemos utilizado antes, el método `children()` devuelve todos los nodos hijos, sea cual sea el nombre del nodo. Es por ello que el objeto `XMLList` resultante contendrá 8 objetos XML: seis nodos `<cancion/>`, un nodo `<titulo/>` y un nodo `<ano/>`.

El siguiente código itera el `XMLList` y pinta en la pantalla de salida los nombres de los nodos gracias al método `name()` de la clase XML

```
var hijos:XMLList = album.children ();
for (var k:int = 0; k < hijos.length (); k++)
```

```
{
trace (hijos [k].name());
}
```

En uno de los ejemplos anteriores hemos realizado una iteración a través de todas las canciones para extraer los títulos de las distintas canciones. El operador descendiente (..) le permite hacer lo mismo.

```
var titulos:XMLList = album..titulo;
```

Este código permite acceder a todos los nodos `<titulo/>` que estén jerárquicamente contenidos dentro del nodo `<album/>` aunque no sean nodos hijo directos. El operador descendiente (..) también puede trabajar en conjunción con el operador atributo (@). De tal forma que podríamos tener expresiones del estilo `album..@id`. Esta expresión devolvería un objeto `XMLList` conteniendo todos aquellos nodos XML con un atributo `id` definido fuere cual fuere la profundidad de anidamiento de esos nodos.

Veamos cómo podríamos pintar el título de todas las canciones así como su duración de una forma sencilla.

```
var titulos:XMLList = album..titulo;
for each (var titulo:XML in titulos)
{
trace (titulo, titulo.parent().@duracion);
}
```

Utilizando la expresión `album..titulo` accederá a los objetos XML que representan los nodos `<titulo/>`. El problema es que la información de la duración de las canciones se encuentra en el nodo padre del nodo `<titulo/>`. Dado un objeto XML se puede acceder a su nodo padre haciendo uso del método `parent()`, el cual devuelve el objeto XML del nodo padre y por lo tanto podrá operar con él como con cualquier otro elemento XML.

## 12.6. Realizando búsquedas

Los operadores que ha visto hasta ahora para recorrer una estructura de un XML son realmente muy cómodos e intuitivos de usar. Pero en casos en los que le pueda interesar obtener las canciones con una duración superior a los 2 minutos, o a las que el autor sea uno en concreto, etc. los operadores vistos no terminarían de ayudarle.

E4X incluye la posibilidad de aplicar filtros directamente con una sintaxis agradable e intuitiva. El siguiente ejemplo muestra cómo localizar aquellas canciones con una duración superior a los 3 minutos (180 segundos).

```
var cancionesLargas:XMLList = album.cancion.(@duracion  
> 180);  
trace (cancionesLargas.length()); // 2
```

Lo único que hemos tenido que hacer es aplicar una expresión que se evalúa en tiempo de ejecución y que se aplica a cada uno de los objetos XML filtrados por `album.cancion`. Si la expresión `@duracion > 180` evalúa a verdadero, el elemento se añade a la lista. Las expresiones pueden ser de complejidades bastante elevadas, incluso permitiendo el uso de Regular Expressions y de múltiples criterios utilizando los operadores `&&` y `||`.

Veamos cómo filtraríamos los datos para conseguir sólo aquellas canciones de "DJ Gere & Mak Attack" con una duración superior a los 180 segundos.

```
var makAttack3:XMLList = album.cancion.(@duracion >  
180).(artista == "DJ Gere & Mak Attack");  
gotoAndPlay("capitulo_13");
```