C
Advanced XML: XPath and XSLT

You've probably already heard about XSL, XSLT, or even XPath. When people talk about XSL they are sometimes confused. These are complicated subjects indeed and it takes time to master them. This appendix provides a very quick introduction to XSLT and XPath, but if you're serious about these technologies, we recommend you buy a separate book or consult additional resources.

What is XSL?

XSL is an XML-based language that became a W3C recommendation at the end of the last decade. It includes three sub-parts:

- XSLT: A language for transforming XML documents
- XPath: A language for accessing and referring any part of an XML document
- XSL-FO: A vocabulary for formatting semantics

W3C defines XSL like this: "Extensible Stylesheet Language (XSL) is a language for expressing stylesheets. It consists of two parts: a language for transforming XML documents, and an XML vocabulary for specifying formatting semantics."

Are you surprised that the definition itself relates XSL to CSS? Well, you shouldn't be, because the name XSL includes 'stylesheet'. While the origins of XSL are in CSS, XSL shares the same functionality and is compatible with CSS2; but there are some differences:

- XSL adds a transforming language called XSLT; while CSS allows you to define font types, backgrounds, and colors to an HTML page. XSLT allows you to transform an XML document into an (X)HTML page, but at the same time, it can also transform an XML document in any other text file format
- XSL adds advanced styling features, grouped under a set of formatting objects and attributes

Because we've talked about CSS and XSL, it is normal to have some questions in this direction:

- "How does XSL differ from CSS?"
- "Is XSL is for XML like CSS is for HTML?"

These two questions come very natural and it's better to have them answered as soon as possible before getting to the implementation part.

How does XSL Differ from CSS?

This question could have been answered partially in one of the paragraphs above, but we will try to make things even clearer. CSS uses the HTML predefined tags to format the document, while XSL uses XML tags defined in the XML document. Another important aspect is how the stylesheets are applied relative to the source tree. With CSS, when applying formatting properties to the source tree they are propagated in the source tree according to its structure so that the properties reflect almost identically into the result tree. With XSL, things are different. Due to the fact that we have a formatting object tree (the XSL document, which itself is an XML document) we can apply the formatting properties differently and thus we might have a different result. The inheritance is done via the formatting object tree and not via the source tree.

Besides these technical details, CSS (1 and 2) implementations are widely available; while for XSL, issues are in an incipient stage.

Is XSL for XML like CSS is for HTML?

CSS can be used for HTML documents, but it can be also used for XML documents. When used with XML documents, it is recommended to have a linear structure in them so that the required transformation will be as simple as possible.

XSL is for XML and it fits very well when heavy manipulation on the XML document is required.

As you can see, it is obvious that CSS and XSL go hand in hand. For example, you can use XSL on the server side for simplifying an XML document and CSS for formatting on the client side.

Most likely, the two will co-exist in the future, because although their domains of applicability intersect, they are not included one into another. They both are very important, and you need to understand them in order to be able to make the best decisions for your projects.

In order to understand how things evolved, let's look back and see how they started.

History

XSL appeared because of the need for a language to handle client-defined XML tags. In September 1998, Microsoft, Texcel, and webMethods submitted a proposal to the W3C under the name of XML Query Language or XQL. In May 1999, the W3C decided to unify the research in this direction under *a common core semantic model for querying* and a result of this was the XSL for Transformation or XSLT**.**

During the development of XSLT, another specification from the XML family was underway: XPointer. XPointer uses the idea of anchor tags in order to allow addressing different parts of a document. While XSLT needed a way for selecting a part of the document in order to transform it, XPointer needed a way to point to specific fragments in the XML document.

The need of selecting different parts of the XML document for different purposes resulted in a common syntax and semantics under the name of XPath. Although XPath is theoretically a subset of XSLT, it can be used independently. Today, XSLT, XPointer (with XLink), and XQuery use XPath to select different parts of an XML document.

After having an overview of XSL, it is time for us to move on and see the specifics of XSLT and XPath. Because XPath can be seen as a separate language, we will start with it.

Setup

This tutorial will use the following XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<data>
  <params>
    <returned_page>1</returned_page>
    <total_pages>6</total_pages>
    <items_count>56</items_count>
    <previous_page></previous_page>
    <next_page>2</next_page>
  </params>
  <grid>
    <row>
      <product_id>1</product_id>
      <name>Santa Costume </name>
      <price>14.99</price>
      <on_promotion>1</on_promotion>
    </row>
    <row>
      <product_id>2</product_id>
      <name>Medieval Lady</name>
      <price>49.99</price>
      <on_promotion>1</on_promotion>
    </row>
    <row>
      <product_id>3</product_id>
      <name>Caveman</name>
      <price>12.99</price>
      <on_promotion>0</on_promotion>
    </row>
    <row>
      <product_id>4</product_id>
      <name>Costume Ghoul</name>
      <price>18.99</price>
      <on_promotion>0</on_promotion> </row>
    <row>
      <product_id>5</product_id>
      <name>Ninja</name>
      <price>15.99</price>
      <on_promotion>0</on_promotion>
    </row>
    <row>
      <product_id>6</product_id>
      <name>Monk</name>
      <price>13.99</price>
      <on_promotion>0</on_promotion>
```

```
        </row>
        <row>
          <product_id>7</product_id>
          <name>Elvis Black Costume</name>
          <price>35.99</price>
          <on_promotion>0</on_promotion>
        </row>
        <row>
          <product_id>8</product_id>
          <name>Robin Hood</name>
          <price>18.99</price>
          <on_promotion>0</on_promotion>
        </row>
        <row>
          <product_id>9</product_id>
          <name>Pierot Clown</name>
          <price>22.99</price>
          <on_promotion>1</on_promotion>
        </row>
        <row>
          <product_id>10</product_id>
          <name>Austin Powers</name>
          <price>49.99</price>
          <on_promotion>0</on_promotion>
        </row>
      </grid>
    </data>
```

The XML document contains data exctracted from the output of the AJAX Grid server script. Each product has an ID, a name, a price, and can be on promotion.

XPath

XPath is a language for retrieving, selecting, and filtering information from an XML document. This information can then be used for different processing tasks. After becoming a W3C recommendation on November 16th, 1999, XPath has been used in different XML-based languages.

XPath offers more than 100 functions for different tasks: node manipulation, date and time functions, number functions, string functions, and more. All these provide an easy way to fetch data from any part of the XML document.

Each XML document is modeled as a tree of nodes. There are seven types of nodes:

- Root nodes
- Element nodes
- Text nodes
- Attribute nodes
- Namespace nodes
- Processing instruction nodes
- Comment nodes

The basic types are:

- Node-set (an unordered collection of nodes without duplicates)
- Boolean (true or false)
- Number (a floating-point number)
- String

The basic syntactic construct of XPath is the expression. The expression is evaluated to an object that can have one of the four basic types that we have seen above. Each expression is evaluated within a given context. This context is given by the language using XPath: XSLT or XPointer. The context can be one of the following:

- A node (the context node)
- A pair of non-zero positive integers (the context position and the context size)
- A set of variable bindings
- A function library
- The set of namespace declarations in scope for the expression

Location paths are the most important type of expressions and therefore we need to present them first. They select a set of nodes in the context node and can also contain expressions that filter the set of nodes that has been selected.

There are two kinds of location paths:

- Absolute paths
- Relative paths

Relative paths consist of several location steps separated by "/". The location paths are interpreted from left to right. Each location step returns a set of nodes that is subsequently used for the next location step as the current context. Absolute paths contain the root node ("/") followed by a relative path.

An example of an absolute path: /step/step/step

And of a relative path: step/step/step

A location step has three sub-parts:

- *An axis*, which specifies the tree relationship between the nodes selected by the location step and the context node
- *A node test*, which specifies the node-type of the nodes selected by the location step
- *Zero or more predicates*, which use arbitrary expressions to further refine the set of nodes selected by the location step

The syntax for a location step contains an axis followed by a double colon, the node test, and zero or more predicates each in square brackets:

```
axis::nodetest[predicate]
```

Or:

```
axis::nodetest[predicate1][predicate2]
```

The following table shows the available axes and their descriptions:

Axis	**Description**
`child`	Contains the children of the context node
`descendant`	Contains the descendants of the context node; a descendant is a child or a child of a child and so on; thus the descendant axis never contains attribute or namespace nodes.
`parent`	Contains the parent of the context node, if there is one.
`ancestor`	Contains the ancestors of the context node; the ancestors of the context node consist of the parent of context node and the parent's parent and so on; thus, the ancestor axis will always include the root node, unless the context node is the root node.
`following-sibling`	Contains all the following siblings of the context node; if the context node is an attribute node or namespace node, the `following-sibling` axis is empty.
`preceding-sibling`	Contains all the preceding siblings of the context node; if the context node is an attribute node or namespace node, the `preceding-sibling` axis is empty.
`following`	Contains all nodes in the same document as the context node that are after the context node in document order, excluding any descendants and excluding attribute nodes and namespace nodes.
`preceding`	Contains all nodes in the same document as the context node that are before the context node in document order, excluding any ancestors and excluding attribute nodes and namespace nodes.
`attribute`	Contains the attributes of the context node; the axis will be empty unless the context node is an element.
`namespace`	Contains the namespace nodes of the context node; the axis will be empty unless the context node is an element.
`self`	Contains just the context node itself.
`descendant-or-self`	Contains the context node and the descendants of the context node.
`ancestor-or-self`	Contains the context node and the ancestors of the context node; thus, the ancestor axis will always include the root node.

XPath Axis

Predicates filter a node-set defined by an axis and produce a new node-set. For each node in the node-set designated by the axis, the predicates are evaluated with that node as the current node. If the predicate is evaluated to true, the node is included in the new node-set, otherwise it is not included.

For location paths, we can also use an abbreviated syntax described in the table below:

Abbreviated syntax	**Description**
`child`	Contains the children of the context node
`*`	Selects all element children of the context node
`@name`	Selects the `name` attribute of the context node
`@*`	Selects all the attributes of the context node
`row[1]`	Selects the first `row` child of the context node
`row[last()]`	Selects the last `row` child of the context node
`*/row`	Selects all `row` grandchildren of the context node
`/grid/row[5]/name[1]`	Selects the first `name` of the fifth `row` of the `grid`
`row//name`	Selects the `name` element descendants of the `row` element children of the context node
`//name`	Selects all the `name` descendants of the document root and thus selects all `name` elements in the same document as the context node
`//row/name`	Selects all the `name` elements in the same document as the context node that have a `row` parent
`.`	Selects the context node
`.//name`	Selects the `name` element descendants of the context node
`row[@name="Caveman"]`	Selects all `row` children of the context node that have a `name` attribute with value `Caveman`
`row[name]`	Selects the `row` children of the context node that have one or more `name` children
`row[@name and @price]`	Selects all the `row` children of the context node that have both a `name` attribute and a `price` attribute

XPath Abbreviated Syntax

As you can see from the table above, the axis `child` is the default one and it can be excluded from the syntax.

In order to complete our short overview of XPath, you need to know which are the available operators and on which operands they can be used, the node functions, number functions, and string functions.

Operator	**Description**	**Result**
`and, or`	Logical and, or	true or false
`= , != , >, >= , <, <=`	Equal, not equal, greater than, greater than or equal to, less than, less than or equal to	true or false
`+,-, *, div, mod`	Plus, minus, multiply, divide, modulus	Number
`|`	Computes two node-sets	A new node-set

XPath Expression Operators

For the functions that can be used on nodes, you can refer to the following table:

Function	Description
`last()`	Returns a number equal to the context size from the expression evaluation context.
`position()`	Returns a number equal to the context position from the expression evaluation context.
`count(node-set)`	Returns the number of nodes in the argument node-set.

XPath Node Functions

For number functions, a short list of functions can be found below:

Function	Description
`sum(node-set)`	Returns the sum, for each node in the argument node-set, of the result of converting the string-values of the node to a number.
`number(object?)`	Converts its argument to a number; If the argument is omitted, it defaults to a node-set with the context node as its only member.
`ceiling(number)`	Returns the smallest (closest to negative infinity) number that is not less than the argument and that is an integer.
`floor(number)`	Returns the largest (closest to positive infinity) number that is not greater than the argument and that is an integer.

XPath Number Functions

For string functions we can use some of those listed below:

Function	Description
`string(object?)`	Converts an object to a string; if the argument is omitted, it defaults to a node-set with the context node as its only member.
`concat(string, string, string*)`	Returns the concatenation of its arguments.
`starts-with(string, string)`	Returns true if the first argument string starts with the second argument string, and otherwise returns false.
`string-length(string?)`	Returns the number of characters in the string; if the argument is omitted, it defaults to the context node converted to a string, in other words the string-value of the context node.
`substring(string, number, number?)`	Returns the substring of the first argument starting at the position specified in the second argument with length specified in the third argument.
`contains(string, string)`	Returns true if the first argument string contains the second argument string, and otherwise returns false.

XPath String Functions

For more on XPath please refer `www.w3.org/TR/xpath` or `www.w3schools.com`.

XSLT

XSLT is used for transforming XML documents. It can transform an XML document to another XML document, or to an HTML document, or even to a text file. After seeing what it's capable of you will probably think twice when you will be facing a problem related to converting an XML file or using it as a data source. It's not that simple, but if you pay attention, you will be able to have in your hands a powerful *weapon* for processing XML documents.

XSLT as a sub-language of XSL is an XML-based language, so we are dealing with the good old XML structure. In order to better understand what we're dealing with, you need to have a clear idea about the meaning of the word *transformation.* Well, transformation refers to the rules of transforming a source XML tree into a result tree. In order to achieve this transformation, we make use of patterns associated with templates. Each pattern is matched against certain elements in the source tree and the template uses these elements to compose the result tree. One important thing to keep in mind is that the source tree and the result tree can differ significantly in terms of structure; the transformations applied to the source tree can reorder, filter, or add new structure elements.

A transformation expressed in XSLT is called a stylesheet. This comes very natural because it defines the way the document is displayed; hence its name.

Because every XSLT document is a well-formed XML document it starts with the usual line:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

After declaring it as an XML document, we need to go further and specify that we're dealing with a stylesheet document. The root element can be one of the two elements: `<xsl:stylesheet>` or `<xsl:transform>`, which tells the parser what version of XSLT we intend to use and what namespace we use for our XSL:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

And:

```
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Note that we are using version 1.0 of the XSLT recommendation and we define a namespace `xsl` that we will use for the rest of the document. The `xsl` namespace will be the prefix for our XSL statements. The namespace allows us to access its attributes, elements, and features.

A stylesheet document contains a set of template rules. Each of these template rules is made of two parts: a pattern that is matched against the nodes of the source tree and a template that can be called to form a part of the result tree. When the XSLT processor receives an XML document that needs to be processed according to a stylesheet document, it looks at every node in the XML document and checks it against the pattern of each template rule in the stylesheet document. If there's a match, the processor outputs the template's rule.

The corresponding element to a template's rule is `xsl:template`. The pattern is specified in the `match` attribute of the `xsl:template` element. The template of the content's output is specified inside the `xsl:template` element. The instructions that do the effective transformation of the source tree into the result tree are XSLT elements. All the elements that do not include the `xsl:` prefix are a part of the result tree.

```
<xsl:template match="expression">
```

This is one of the base XSLT elements for a sylesheet document. In order to match the entire document, we will use `match="/"`. Another important attribute is `name` which can be used for giving a name to our template in case we need to call it as we will see later in this chapter.

Another element that is very useful is:

```
<xsl:for-each select="expression">
```

This element goes through all the nodes returned by the XPath expression contained in the `select` attribute. It can be easily compared to a `for` instruction in PHP, where the select attribute can be the condition for which the `for` body is executed. By putting `select='data/grid/row'` we select all the rows in our XML that contain the products.

```
<xsl:value-of select="expression"/>
```

The `xsl:value-of` element extracts the value from the selected node and copies it to the output. For example:

```
<xsl:for-each select="data/grid/row">
<xsl:value-of select="name"/>
</xsl:for-each>
```

would output the names of all the products in our XML source file.

```
xsl:attribute
```

The `xsl:attribute` is used for defining an attribute to an element. For example

```
<img><xsl:attribute name="src"><xsl:value-of
select="src"/></xsl:attribute></img>
```

would add the `src` attribute to the HTML element `img` and set its value to the `src` value of the current product. In order for this example to work we would have defined each product in our initial XML like this:

```
<row>
  <product_id>1</product_id>
  <name>Santa Costume </name>
  <price>14.99</price>
  <on_promotion>1</on_promotion>
  <src>picture.jpg</src>
</row>
```

If we've talked about adding an attribute to an element, we also need to know that we can define our own element.

```
<xsl:element name="element-name">
```

The `xsl:element` can be useful if we need to define our own node in the result tree. The `name` attribute specifies its name and this name will be displayed as output.

```
<xsl:if test="expression">
```

The `xsl:if` element is used to test a condition against the nodes of the source tree and outputs something only if the test boolean condition is evaluated to true.

```
<xsl:if test=" on_promotion &gt; 0">
  <input type="checkbox" name="on_promotion" disabled="disabled"
checked="checked"/>
</xsl:if>
```

The example just given outputs a checked checkbox only if the product is on promotion. In order to have something if the test condition is evaluated to false, we need to use another `xsl:if` element or an `xsl:choose` element.

xsl:choose / xsl:when / xsl:otherwise

Because sometimes analogy works the best, `xsl:choose / xsl:when / xsl: otherwise` implement the same functionality as a `switch/case/default` in PHP or C++. Here's an example:

```
<xsl:choose>
  <xsl:when test="on_promotion &gt; 0">
    <input type="checkbox" name="on_promotion" disabled="disabled"
           checked="checked"/>
  </xsl:when>
  <xsl:otherwise>
    <input type="checkbox" name="on_promotion" disabled="disabled" />
  </xsl:otherwise>
</xsl:choose>
```

The example above implements the same functionality as an `if` statement, so it is the simplest example we could imagine taking in consideration `xsl:choose` is the alternative for multiple `xsl:if` elements. It outputs a checked checkbox if the product is on promotion or an unchecked one otherwise.

<xsl:call-template name="template-name">

The `xsl:call-template` element calls a named template:

```
<xsl:template match="/data/grid/row" name="row">
...
</xsl:template>
<xsl:call-template name="row"/>
```

The example above would call a template named `row` for all the rows in our initial XML file.

More on XSLT can be found at `www.w3.org/TR/xslt`, `www.w3schools.com`, or `www.topxml.com`.

Testing XPath and XSLT

Because modern web browsers include XPath and XSLT support, you can use your browser to test your knowledge of XSL.

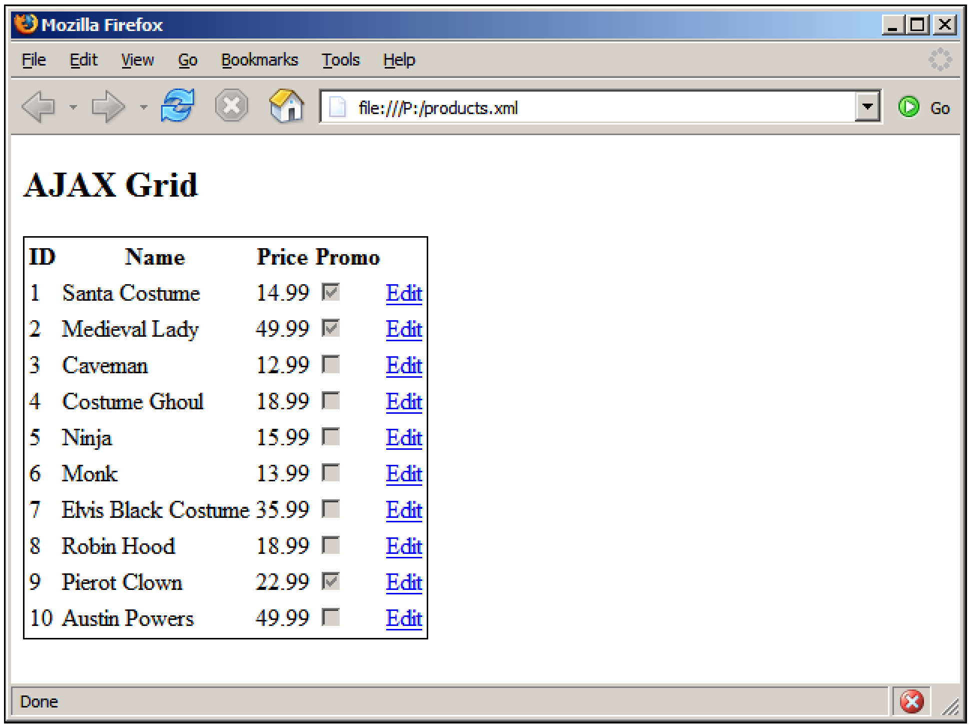
To begin with, you need to create an XML file that references an XSL file. Please create a new file named `products.xml`. (This code is the same as the code in the *Setup* section of this appendix except for the highlighted line.) Add the following code:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="products.xsl"?>
<data>
  <params>
    <returned_page>1</returned_page>
    <total_pages>6</total_pages>
    <items_count>56</items_count>
    <previous_page></previous_page>
    <next_page>2</next_page>
  </params>
  <grid>
    <row>
      <product_id>1</product_id>
      <name>Santa Costume </name>
      <price>14.99</price>
      <on_promotion>1</on_promotion>
    </row>
    <row>
      <product_id>2</product_id>
      <name>Medieval Lady</name>
      <price>49.99</price>
      <on_promotion>1</on_promotion>
    </row>
    <row>
      <product_id>3</product_id>
      <name>Caveman</name>
      <price>12.99</price>
      <on_promotion>0</on_promotion>
    </row>
    <row>
      <product_id>4</product_id>
      <name>Costume Ghoul</name>
      <price>18.99</price>
      <on_promotion>0</on_promotion> </row>
    <row>
      <product_id>5</product_id>
      <name>Ninja</name>
      <price>15.99</price>
      <on_promotion>0</on_promotion>
    </row>
    <row>
      <product_id>6</product_id>
      <name>Monk</name>
      <price>13.99</price>
      <on_promotion>0</on_promotion>
    </row>
    <row>
      <product_id>7</product_id>
      <name>Elvis Black Costume</name>
      <price>35.99</price>
      <on_promotion>0</on_promotion>
    </row>
    <row>
      <product_id>8</product_id>
      <name>Robin Hood</name>
      <price>18.99</price>
      <on_promotion>0</on_promotion>
    </row>
    <row>
      <product_id>9</product_id>
      <name>Pierot Clown</name>
      <price>22.99</price>
      <on_promotion>1</on_promotion>
    </row>
    <row>
      <product_id>10</product_id>
      <name>Austin Powers</name>
      <price>49.99</price>
      <on_promotion>0</on_promotion>
    </row>
  </grid>
</data>
```

Then create, in the same folder, a file named `products.xsl`, with this code:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
    <body>
    <h2>AJAX Grid</h2>
      <table style="border:1px solid black;">
        <tr>
          <th>ID</th>
          <th>Name</th>
          <th>Price</th>
          <th>Promo</th>
        </tr>
        <xsl:for-each select="data/grid/row">
          <xsl:element name="tr">
            <xsl:attribute name="id">
              <xsl:value-of select="product_id" />
            </xsl:attribute>
            <td><xsl:value-of select="product_id" /></td>
            <td><xsl:value-of select="name" /> </td>
            <td><xsl:value-of select="price" /></td>
            <td>
              <xsl:choose>
                <xsl:when test="on_promotion &gt; 0">
                  <input type="checkbox" name="on_promotion"
                         disabled="disabled" checked="checked"/>
                </xsl:when>
                <xsl:otherwise>
                  <input type="checkbox" name="on_promotion"
                         disabled="disabled"/>
                </xsl:otherwise>
               </xsl:choose>
            </td>
            <td>
              <xsl:element name="a">
                <xsl:attribute name = "href">#</xsl:attribute>
                Edit
              </xsl:element>
            </td>
          </xsl:element>
        </xsl:for-each>
      </table>
    </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Loading `products.xml` in a web browser such as Internet Explorer or Firefox will show you the XSLT-formatted list of products, as shown in Figure C.1:



AJAX Grid

ID	Name	Price	Promo	
1	Santa Costume	14.99	✓	Edit
2	Medieval Lady	49.99	✓	Edit
3	Caveman	12.99	☐	Edit
4	Costume Ghoul	18.99	☐	Edit
5	Ninja	15.99	☐	Edit
6	Monk	13.99	☐	Edit
7	Elvis Black Costume	35.99	☐	Edit
8	Robin Hood	18.99	☐	Edit
9	Pierot Clown	22.99	✓	Edit
10	Austin Powers	49.99	☐	Edit

Figure C.1: Transformed XML