XML Expressions and filters

Hans Hagen

1. Path expressions

In ConTEXt we use LPATH expressions, which are a variant on xpath expressions as in XSLT but in this case more geared towards usage in TEX. This mechanism will be extended when demands are there.
A path is a sequence of matches. A simple path expression is:

```
a/b/c/d
```

Here each / goes one level deeper. We can go backwards in a lookup with ..:

```
a/b/../d
```

We can also combine lookups, as in:

```
a/(b|c)/d
```

A negated lookup is preceded by a !:

```
a/(b|c)/!d
```

A wildcard is specified with a *:

```
a/(b|c)/!d/e/*/f
```

In addition to these tag based lookups we can use attributes:

```
a/(b|c)/!d/e/*/f[@type=whatever]
```

An @ as first character means that we are dealing with an attribute. Within the square brackets there can be boolean expressions:

```
a/(b|c)/!d/e/*/f[@type=whatever and @id>100]
```

You can use functions as in:

```
a/(b|c)/!d/e/*/f[something(text()) == "oeps"]
```

There are a couple of predefined functions:

`rootposition` `order`	number	the index of the matched root element (kind of special)
`position`	number	the current index of the matched element in the match list
`match`	number	the current index of the matched element sub list with the same parent
`first`	number	
`last`	number	
`index`	number	the current index of the matched element in its parent list
`firstindex`	number	
`lastindex`	number	
`element`	number	the element's index
`firstelement`	number	
`lastelement`	number	
`text`	string	the textual representation of the matched element
`content`	table	the node of the matched element
`name`	string	the full name of the matched element: namespace and tag
`namespace` `ns`	string	the namespace of the matched element
`tag`	string	the tag of the matched element
`attribute`	string	the value of the attribute with the given name of the matched element

There are fundamental differences between `position`, `match` and `index`. Each step results in a new list of matches. The `position` is the index in this new (possibly intermediate) list. The `match` is also an index in this list but related to the specific match of element names. The `index` refers to the location in the parent element.
Say that we have:

```
<collection>
  <resources>
    <manual>
      <screen>.1.</screen>
```

```
      <paper>.1.</paper>
    </manual>
    <manual>
      <paper>.2.</paper>
      <screen>.2.</screen>
    </manual>
  <resources>
  <resources>
    <manual>
      <screen>.3.</screen>
      <paper>.3.</paper>
    </manual>
  <resources>
<collection>
```

The following then applies:

```
collection/resources/manual[position()==1]/paper  .1.
collection/resources/manual[match()==1]/paper     .1. .3.
collection/resources/manual/paper[index()==1]     .2.
```

In most cases the `position` test is more restrictive than the `match` test.
You can pass your own functions too. Such functions are defined in the `xml.expressions` namespace. We have defined a few shortcuts:

`find(str,pattern)`	`string.find`
`contains(str)`	`string.find`
`oneof(str,...)`	is `str` in list
`upper(str)`	`characters.upper`
`lower(str)`	`characters.lower`
`number(str)`	`tonumber`
`boolean(str)`	`toboolean`
`idstring(str)`	removes leading hash
`name(index)`	full tag name
`tag(index)`	tag name
`namespace(index)`	namespace of tag
`text(index)`	content
`error(str)`	quit and show error
`quit()`	quit
`print()`	print message
`count(pattern)`	number of matches
`child(pattern)`	take child that matches

You can also use normal Lua functions as long as you make sure that you pass the right arguments. There are a few predefined variables available inside such functions.

`list`	table	the list of matches
`l`	number	the current index in the list of matches
`ll`	element	the current element that matched
`order`	number	the position of the root of the path

The given expression between `[]` is converted to a Lua expression so you can use the usual operators:

```
== ~= <= >= < > not and or ()
```

In addition, = equals == and != is the same as ~=. If you mess up the expression, you quite likely get a Lua error message.

2. CSS selectors

The CSS approach to filtering is a bit different from the path based one and is supported too. In fact, you can combine both methods. Depending on what you select, the CSS one can be a little bit faster too. It has the advantage that one can select more in one go but at the same time looks a bit less attractive. This method was added just to show that it can be done but might be useful too. A selector is given between curly braces (after all CSS uses them and they have no function yet in the parser).

```
\xmlall{#1}{{foo bar .whatever, bar foo .whatever}}
```

The following methods are supported:

`element`	all tags element
`element-1 > element-2`	all tags element-2 with parent tag element-1
`element-1 + element-2`	all tags element-2 preceded by tag element-1
`element-1 ~ element-2`	all tags element-2 preceded by tag element-1
`element-1 element-2`	all tags element-2 inside tag element-1
`[attribute]`	has attribute
`[attribute=value]`	attribute equals value
`[attribute~=value]`	attribute contains value (space is separator)
`[attribute^="value"]`	attribute starts with value
`[attribute$="value"]`	attribute ends with value
`[attribute*="value"]`	attribute contains value
`.class`	has class
`#id`	has id
`:nth-child(n)`	the child at index n
`:nth-last-child(n)`	the child at index n from the end
`:first-child`	the first child
`:last-child`	the last child
`:nth-of-type(n)`	the match at index n

`:nth-last-of-type(n)`	the match at index n from the end
`:first-of-type`	the first match
`:last-of-type`	the last match
`:only-of-type`	the only match or nothing
`:only-child`	the only child or nothing
`:empty`	only when empty
`:root`	the whole tree

The next pages show some examples. For that we use the demo file:

```
<?xml version="1.0" ?>

<a>
    <b class="one">b.one</b>
    <b class="two">b.two</b>
    <b class="one two">b.one.two</b>
    <b class="three">b.three</b>
    <b id="first">b#first</b>
    <c>c</c>
    <d>d e</d>
    <e>d e</e>
    <e>d e e</e>
    <d>d f</d>
    <f foo="bar">@foo = bar</f>
    <f bar="foo">@bar = foo</f>
    <f bar="foo1">@bar = foo1</f>
    <f bar="foo2">@bar = foo2</f>
    <f bar="foo3">@bar = foo3</f>
    <f bar="foo+4">@bar = foo+4</f>
    <g>g</g>
    <g><gg><d>g gg d</d></gg></g>
    <g><gg><f>g gg f</f></gg></g>
    <g><gg><f class="one">g gg f.one</f></gg></g>
    <g>g</g>
    <g><gg><f class="two">g gg f.two</f></gg></g>
    <g><gg><f class="three">g gg f.three</f></gg></g>
    <g><f class="one">g f.one</f></g>
    <g><f class="three">g f.three</f></g>
    <h whatever="four five six">@whatever = four five six</h>
</a>
```

The class and id selectors often only make sense in HTML like documents but they are supported nevertheless. They are after all just shortcuts for filtering by attribute. The class filtering is special in the sense that it checks for a class in a list of classes given in an attribute.

```
——— .one ———————————————————————————————————————————————

1  b.one
2  b.one.two
3  g gg f.one
4  g f.one

——— .one, .two ——————————————————————————————————————————

1  b.one
2  b.two
3  b.one.two
4  g gg f.one
5  g gg f.two
6  g f.one

——— .one, .two, #first ——————————————————————————————————

1  b.one
2  b.two
3  b.one.two
4  b#first
5  g gg f.one
6  g gg f.two
7  g f.one
```

Attributes can be filtered by presence, value, partial value and such. Quotes are optional but we advise to use them.

```
——— [foo], [bar=foo] ————————————————————————————————————

1  @foo = bar
2  @bar = foo

——— [bar~=foo] ——————————————————————————————————————————

1  @bar = foo

——— [bar^="foo"] ————————————————————————————————————————

1  @bar = foo
2  @bar = foo1
3  @bar = foo2
4  @bar = foo3
5  @bar = foo+4

——— [whatever~="five"] ——————————————————————————————————

1  @whatever = four five six
```

You can of course combine the methods as in:

```
——— g f .one, g f .three ———————————————————————————————

1  g gg f.one
2  g gg f.three
3  g f.one
4  g f.three

——— g > f .one, g > f .three ———————————————————————————

1  g f.one
2  g f.three

——— d + e ——————————————————————————————————————————————

1  d e

——— d ~ e ——————————————————————————————————————————————

1  d e
2  d e e

——— d ~ e, g f .one, g f .three ————————————————————————

1  d e
2  d e e
3  g gg f.one
4  g gg f.three
5  g f.one
6  g f.three
```

You can also negate the result by using `:not` on a simple expression:

```
——— :not([whatever~="five"]) ———————————————————————————

1  <?xml version="1.0" ?>
   <a>
       <b class="one">b.one</b>
       <b class="two">b.two</b>
       <b class="one two">b.one.two</b>
       <b class="three">b.three</b>
       <b id="first">b#first</b>
       <c>c</c>
       <d>d e</d>
       <e>d e</e>
       <e>d e e</e>
       <d>d f</d>
       <f foo="bar">@foo = bar</f>
```

```
        <f bar="foo">@bar = foo</f>
        <f bar="foo1">@bar = foo1</f>
        <f bar="foo2">@bar = foo2</f>
        <f bar="foo3">@bar = foo3</f>
        <f bar="foo+4">@bar = foo+4</f>
        <g>g</g>
        <g><gg><d>g gg d</d></gg></g>
        <g><gg><f>g gg f</f></gg></g>
        <g><gg><f class="one">g gg f.one</f></gg></g>
        <g>g</g>
        <g><gg><f class="two">g gg f.two</f></gg></g>
        <g><gg><f class="three">g gg f.three</f></gg></g>
        <g><f class="one">g f.one</f></g>
        <g><f class="three">g f.three</f></g>
        <h whatever="four five six">@whatever = four five six</h>
    </a>
2
        <b class="one">b.one</b>
        <b class="two">b.two</b>
        <b class="one two">b.one.two</b>
        <b class="three">b.three</b>
        <b id="first">b#first</b>
        <c>c</c>
        <d>d e</d>
        <e>d e</e>
        <e>d e e</e>
        <d>d f</d>
        <f foo="bar">@foo = bar</f>
        <f bar="foo">@bar = foo</f>
        <f bar="foo1">@bar = foo1</f>
        <f bar="foo2">@bar = foo2</f>
        <f bar="foo3">@bar = foo3</f>
        <f bar="foo+4">@bar = foo+4</f>
        <g>g</g>
        <g><gg><d>g gg d</d></gg></g>
        <g><gg><f>g gg f</f></gg></g>
        <g><gg><f class="one">g gg f.one</f></gg></g>
        <g>g</g>
        <g><gg><f class="two">g gg f.two</f></gg></g>
        <g><gg><f class="three">g gg f.three</f></gg></g>
        <g><f class="one">g f.one</f></g>
        <g><f class="three">g f.three</f></g>
        <h whatever="four five six">@whatever = four five six</h>
3   b.one
4   b.two
5   b.one.two
6   b.three
7   b#first
8   c
```

```
9  d e
10 d e
11 d e e
12 d f
13 @foo = bar
14 @bar = foo
15 @bar = foo1
16 @bar = foo2
17 @bar = foo3
18 @bar = foo+4
19 g
20 <gg><d>g gg d</d></gg>
21 <d>g gg d</d>
22 g gg d
23 <gg><f>g gg f</f></gg>
24 <f>g gg f</f>
25 g gg f
26 <gg><f class="one">g gg f.one</f></gg>
27 <f class="one">g gg f.one</f>
28 g gg f.one
29 g
30 <gg><f class="two">g gg f.two</f></gg>
31 <f class="two">g gg f.two</f>
32 g gg f.two
33 <gg><f class="three">g gg f.three</f></gg>
34 <f class="three">g gg f.three</f>
35 g gg f.three
36 <f class="one">g f.one</f>
37 g f.one
38 <f class="three">g f.three</f>
39 g f.three
```

——— :not(d) ———————————————————————————————————————

```
1
       <b class="one">b.one</b>
       <b class="two">b.two</b>
       <b class="one two">b.one.two</b>
       <b class="three">b.three</b>
       <b id="first">b#first</b>
       <c>c</c>
       <d>d e</d>
       <e>d e</e>
       <e>d e e</e>
       <d>d f</d>
       <f foo="bar">@foo = bar</f>
       <f bar="foo">@bar = foo</f>
       <f bar="foo1">@bar = foo1</f>
       <f bar="foo2">@bar = foo2</f>
```

```
       <f bar="foo3">@bar = foo3</f>
       <f bar="foo+4">@bar = foo+4</f>
       <g>g</g>
       <g><gg><d>g gg d</d></gg></g>
       <g><gg><f>g gg f</f></gg></g>
       <g><gg><f class="one">g gg f.one</f></gg></g>
       <g>g</g>
       <g><gg><f class="two">g gg f.two</f></gg></g>
       <g><gg><f class="three">g gg f.three</f></gg></g>
       <g><f class="one">g f.one</f></g>
       <g><f class="three">g f.three</f></g>
       <h whatever="four five six">@whatever = four five six</h>
2  b.one
3  b.two
4  b.one.two
5  b.three
6  b#first
7  c
8  d e
9  d e e
10 @foo = bar
11 @bar = foo
12 @bar = foo1
13 @bar = foo2
14 @bar = foo3
15 @bar = foo+4
16 g
17 <gg><d>g gg d</d></gg>
18 <d>g gg d</d>
19 <gg><f>g gg f</f></gg>
20 <f>g gg f</f>
21 g gg f
22 <gg><f class="one">g gg f.one</f></gg>
23 <f class="one">g gg f.one</f>
24 g gg f.one
25 g
26 <gg><f class="two">g gg f.two</f></gg>
27 <f class="two">g gg f.two</f>
28 g gg f.two
29 <gg><f class="three">g gg f.three</f></gg>
30 <f class="three">g gg f.three</f>
31 g gg f.three
32 <f class="one">g f.one</f>
33 g f.one
34 <f class="three">g f.three</f>
35 g f.three
36 @whatever = four five six
```

The child and match selectors are also supported:

```
—— a:nth-child(3) ——————————————————————————————————————————

1  b.one.two

—— a:nth-last-child(3) ———————————————————————————————————————

1  <f class="one">g f.one</f>

—— g:nth-of-type(3) ——————————————————————————————————————————

1  <gg><f>g gg f</f></gg>

—— g:nth-last-of-type(3) ——————————————————————————————————————

1  <gg><f class="three">g gg f.three</f></gg>

—— a:first-child ——————————————————————————————————————————

1  b.one

—— a:last-child ———————————————————————————————————————————

1  @whatever = four five six

—— e:first-of-type ————————————————————————————————————————

1  d e

—— gg d:only-of-type ——————————————————————————————————————

1  g gg d
```

Instead of numbers you can also give the **an** and **an+b** formulas as well as the **odd** and **even** keywords:

```
—— a:nth-child(even) ——————————————————————————————————————

1  b.two
2  b.three
3  c
4  d e
5  d f
6  @bar = foo
7  @bar = foo2
8  @bar = foo+4
9  <gg><d>g gg d</d></gg>
10 <gg><f class="one">g gg f.one</f></gg>
```

```
11 <gg><f class="two">g gg f.two</f></gg>
12 <f class="one">g f.one</f>
```

—— a:nth-child(odd) ——————————————————————————————

```
1  b.one
2  b.one.two
3  b#first
4  d e
5  d e e
6  @foo = bar
7  @bar = foo1
8  @bar = foo3
9  g
10 <gg><f>g gg f</f></gg>
11 g
12 <gg><f class="three">g gg f.three</f></gg>
13 <f class="three">g f.three</f>
```

—— a:nth-child(3n+1) ——————————————————————————————

```
1  b.one
2  b.two
3  b.one.two
4  b.three
5  b#first
6  c
7  d e
8  d e
9  d e e
10 d f
11 @foo = bar
12 @bar = foo
13 @bar = foo1
14 @bar = foo2
15 @bar = foo3
16 @bar = foo+4
17 g
18 <gg><d>g gg d</d></gg>
19 <gg><f>g gg f</f></gg>
20 <gg><f class="one">g gg f.one</f></gg>
21 g
22 <gg><f class="two">g gg f.two</f></gg>
23 <gg><f class="three">g gg f.three</f></gg>
```

—— a:nth-child(2n+3) ——————————————————————————————

```
1  b.one.two
2  c
```

```
3  d e e
4  @bar = foo
5  @bar = foo3
6  <gg><d>g gg d</d></gg>
7  g
8  <f class="one">g f.one</f>
```

There are a few special cases:

```
─── g:empty ──────────────────────────────────────────────────────────────

─── g:root ───────────────────────────────────────────────────────────────

1
       <b class="one">b.one</b>
       <b class="two">b.two</b>
       <b class="one two">b.one.two</b>
       <b class="three">b.three</b>
       <b id="first">b#first</b>
       <c>c</c>
       <d>d e</d>
       <e>d e</e>
       <e>d e e</e>
       <d>d f</d>
       <f foo="bar">@foo = bar</f>
       <f bar="foo">@bar = foo</f>
       <f bar="foo1">@bar = foo1</f>
       <f bar="foo2">@bar = foo2</f>
       <f bar="foo3">@bar = foo3</f>
       <f bar="foo+4">@bar = foo+4</f>
       <g>g</g>
       <g><gg><d>g gg d</d></gg></g>
       <g><gg><f>g gg f</f></gg></g>
       <g><gg><f class="one">g gg f.one</f></gg></g>
       <g>g</g>
       <g><gg><f class="two">g gg f.two</f></gg></g>
       <g><gg><f class="three">g gg f.three</f></gg></g>
       <g><f class="one">g f.one</f></g>
       <g><f class="three">g f.three</f></g>
       <h whatever="four five six">@whatever = four five six</h>

─── * ────────────────────────────────────────────────────────────────────

1
       <b class="one">b.one</b>
       <b class="two">b.two</b>
       <b class="one two">b.one.two</b>
       <b class="three">b.three</b>
       <b id="first">b#first</b>
```

```
        <c>c</c>
        <d>d e</d>
        <e>d e</e>
        <e>d e e</e>
        <d>d f</d>
        <f foo="bar">@foo = bar</f>
        <f bar="foo">@bar = foo</f>
        <f bar="foo1">@bar = foo1</f>
        <f bar="foo2">@bar = foo2</f>
        <f bar="foo3">@bar = foo3</f>
        <f bar="foo+4">@bar = foo+4</f>
        <g>g</g>
        <g><gg><d>g gg d</d></gg></g>
        <g><gg><f>g gg f</f></gg></g>
        <g><gg><f class="one">g gg f.one</f></gg></g>
        <g>g</g>
        <g><gg><f class="two">g gg f.two</f></gg></g>
        <g><gg><f class="three">g gg f.three</f></gg></g>
        <g><f class="one">g f.one</f></g>
        <g><f class="three">g f.three</f></g>
        <h whatever="four five six">@whatever = four five six</h>
2  b.one
3  b.two
4  b.one.two
5  b.three
6  b#first
7  c
8  d e
9  d e
10 d e e
11 d f
12 @foo = bar
13 @bar = foo
14 @bar = foo1
15 @bar = foo2
16 @bar = foo3
17 @bar = foo+4
18 g
19 <gg><d>g gg d</d></gg>
20 <d>g gg d</d>
21 g gg d
22 <gg><f>g gg f</f></gg>
23 <f>g gg f</f>
24 g gg f
25 <gg><f class="one">g gg f.one</f></gg>
26 <f class="one">g gg f.one</f>
27 g gg f.one
28 g
29 <gg><f class="two">g gg f.two</f></gg>
```

```
30 <f class="two">g gg f.two</f>
31 g gg f.two
32 <gg><f class="three">g gg f.three</f></gg>
33 <f class="three">g gg f.three</f>
34 g gg f.three
35 <f class="one">g f.one</f>
36 g f.one
37 <f class="three">g f.three</f>
38 g f.three
39 @whatever = four five six
```

Combining the CSS methods with the regular ones is possible:

```
—— g gg f .one ——————————————————————————————————————————

1  g gg f.one

—— g/gg/f[@class='one'] ——————————————————————————————————

1  g gg f.one

—— g/gg f .one ——————————————————————————————————————————

1  g gg f.one
```

The next examples we use this file:

```
<?xml version="1.0" ?>

<document>
    <title class="one"  >title 1</title>
    <title class="two"  >title 2</title>
    <title class="one"  >title 3</title>
    <title class="three">title 4</title>
</document>
```

When we filter from this (not too well structured) tree we can use both methods to achieve the same:

```
—— document title .one, document title .three ——————————————

1  title 1
2  title 3
3  title 4
```

```
——— /document/title[(@class='one') or (@class='three')] ————————————
```

```
1  title 1
2  title 3
3  title 4
```

However, imagine this file:

```
<?xml version="1.0" ?>

<document>
    <title    class="one">title 1</title>
    <subtitle class="sub">title 1.1</subtitle>
    <title    class="two">title 2</title>
    <subtitle class="sub">title 2.1</subtitle>
    <title    class="one">title 3</title>
    <subtitle class="sub">title 3.1</subtitle>
    <title    class="two">title 4</title>
    <subtitle class="sub">title 4.1</subtitle>
</document>
```

The next filter is easier with the CSS selector methods because these accumulate independent (simple) expressions:

```
——— document title .one + subtitle, document title .two + subtitle –
```

```
1  title 1.1
2  title 2.1
3  title 3.1
4  title 4.1
```

Watch how we get an output in the document order. Because we render a sequential document a combined filter will trigger a sorting pass.

3. Functions as filters

At the Lua end a whole LPATH expression results in a (set of) node(s) with its environment, but that is hardly usable in TEX. Think of code like:

```
for e in xml.collected(xml.load('text.xml'),"title") do
  -- e = the element that matched
end
```

The older variant is still supported but you can best use the previous variant.

```
for r, d, k in xml.elements(xml.load('text.xml'),"title") do
  -- r = root of the title element
  -- d = data table
  -- k = index in data table
end
```

Here `d[k]` points to the `title` element and in this case all titles in the tree pass by. In practice this kind of code is encapsulated in function calls, like those returning elements one by one, or returning the first or last match. The result is then fed back into TEX, possibly after being altered by an associated setup.
In addition to the previously discussed expressions, one can add so called filters to the expression, for instance:

```
a/(b|c)/!d/e/text()
```

In a filter, the last part of the LPATH expression is a function call. The previous example returns the text of each element `e` that results from matching the expression. When running TEX the following functions are available. Some are also available when using pure Lua. In TEX you can often use one of the macros like `\xmlfirst` instead of a `\xmlfilter` with finalizer `first()`. The filter can be somewhat faster but that is hardly noticeable.

`context()`	string	the serialized text with TEX catcode regime
`function()`	string	depends on the function
`name()`	string	the (remapped) namespace
`tag()`	string	the name of the element
`tags()`	list	the names of the element
`text()`	string	the serialized text
`upper()`	string	the serialized text uppercased
`lower()`	string	the serialized text lowercased
`stripped()`	string	the serialized text stripped
`lettered()`	string	the serialized text only letters (cf. UNICODE)
`count()`	number	the number of matches
`index()`	number	the matched index in the current path
`match()`	number	the matched index in the preceding path
`attribute(name)`	content	returns the attribute with the given name
`chainattribute(name)`	content	idem, but backtracks till one is found
`command(name)`	content	expands the setup with the given name for each found element
`position(n)`	content	processes the n^{th} instance of the found element
`all()`	content	processes all instances of the found element
`reverse()`	content	idem in reverse order
`first()`	content	processes the first instance of the found element

`last()`	content	processes the last instance of the found element
`concat(...)`	content	concatenates the match
`concatrange(from,to,...)`	content	concatenates a range of matches

The extra arguments of the concatenators are: `separator` (string), `lastseparator` (string) and `textonly` (a boolean).
These filters are in fact Lua functions which means that if needed more of them can be added. Indeed this happens in some of the XML related MkIV modules, for instance in the MATHML processor.

4. Example

The number of commands is rather large and if you want to avoid them this is often possible. Take for instance:

```
\xmlall{#1}{/a/b[position()>3]}
```

Alternatively you can use:

```
\xmlfilter{#1}{/a/b[position()>3]/all()}
```

and actually this is also faster as internally it avoids a function call. Of course in practice this is hardly measurable.
In previous examples we've already seen quite some expressions, and it might be good to point out that the syntax is modeled after XSLT but is not quite the same. The reason is that we started with a rather minimal system and have already styles in use that depend on compatibility.

```
namespace:// axis node(set) [expr 1]..[expr n] / ... / filter
```

When we are inside a ConTEXt run, the namespace is `tex`. However, if you want not to print back to TEX you need to be more explicit. Say that we typeset exams and have a (not that logical) structure like:

```
<question>
  <text>...</text>
  <answer>
```

```
    <item>one</item>
    <item>two</item>
    <item>three</item>
  </answer>
  <alternative>
    <condition>true</condition>
    <score>1</score>
  </alternative>
  <alternative>
    <condition>false</condition>
    <score>0</score>
  </alternative>
  <alternative>
    <condition>true</condition>
    <score>2</score>
  </alternative>
</question>
```

Say that we typeset the questions with:

```
\startxmlsetups question
  \blank
  score: \xmlfunction{#1}{totalscore}
  \blank
  \xmlfirst{#1}{text}
  \startitemize
      \xmlfilter{#1}{/answer/item/command(answer:item)}
  \stopitemize
  \endgraf
  \blank
\stopxmlsetups
```

Each item in the answer results in a call to:

```
\startxmlsetups answer:item
  \startitem
    \xmlflush{#1}
    \endgraf
    \xmlfilter{#1}{../../alternative[position()=rootposition()]/
      condition/command(answer:condition)}
  \stopitem
\stopxmlsetups
```

```
\startxmlsetups answer:condition
  \endgraf
  condition: \xmlflush{#1}
  \endgraf
\stopxmlsetups
```

Now, there are two rather special filters here. The first one involves calculating the total score. As we look forward we use a function to deal with this.

```
\startluacode
function xml.functions.totalscore(root)
  local score = 0
  for e in xml.collected(root,"/alternative") do
    score = score + xml.filter(e,"xml:///score/number()") or 0
  end
  tex.write(score)
end
\stopluacode
```

Watch how we use the namespace to keep the results at the Lua end.
The second special trick shown here is to limit a match using the current position of the root (#) match.
As you can see, a path expression can be more than just filtering a few nodes. At the end of this manual you will find a bunch of examples.

5. Tables

If you want to know how the internal XML tables look you can print such a table:

```
print(table.serialize(e))
```

This produces for instance:

```
t={
 ["at"]={
  ["label"]="whatever",
 },
 ["dt"]={ "some text" },
 ["ns"]="",
 ["rn"]="",
```

```
 ["tg"]="demo",
}
```

The `rn` entry is the renamed namespace (when renaming is applied). If you see tags like `@pi@` this means that we don't have an element, but (in this case) a processing instruction.

`@rt@` the root element
`@dd@` document definition
`@cm@` comment, like `<!-- whatever -->`
`@cd@` so called `CDATA`
`@pi@` processing instruction, like `<?whatever we want ?>`

There are many ways to deal with the content, but in the perspective of TEX only a few matter.

`xml.sprint(e)` print the content to TEX and apply setups if needed
`xml.tprint(e)` print the content to TEX (serialize elements verbose)
`xml.cprint(e)` print the content to TEX (used for special content)

Keep in mind that anything low level that you uncover is not part of the official interface unless mentioned in this manual.