

Extending XQuery with pattern matching: Motivation

Regular expressions are a standard way to validate/query strings.

Example

`fooba+r`

matches `foobar`
or `foobaar`
or `foobaaar`
but not `foob̄r`

Example

`(.*)://([~/]+)(/(.*))?`

matches `https://example.org`

or `http://balisage.net/2014/Program.html`

The diagram shows the regular expression `(.*)://([~/]+)(/(.*))?` with four numbered groups indicated by curly braces and numbers below them:
1: `(.*)`
2: `//`
3: `([~/]+)`
4: `(/(.*))?`

matches `https://example.org`
1: `https`, 2: `://example.org`

or `http://balisage.net/2014/Program.html`
1: `http`, 2: `://balisage.net`, 3: `/2014/Program.html`, 4: `/`

Can we use something similar to validate/query XML?

Regular expressions are problematic there:

Example

`<xml>(.*)</xml>`

matches `<xml>data</xml> :`
but also `<xml>data</xml><!--</xml>-> :(`

We need regular expressions that understand the XML structure.

Extending XQuery with pattern matching: Motivation

But XPath is great. Why would we need regular expressions?

Example

No, it is really bad for siblings, as they often are on webpages:

```
<html>
  <h2>Section start</h2>
  <p>data we want </p>
  <p>continued</p>
  <h2>Section end</h2>
</html>
```

```
/html/h2[. eq 'Section start']/following-sibling::p intersect
/html/h2[. eq 'Section end']/preceding-sibling::p
```

```
<html>
  <h2>Section start</h2>
  (<p>.*</p>)*
  <h2>Section end</h2>
</html>
```

And XPath does not check for input errors
The above queries applied to

```
<xml>something entirely different</xml>
```

Query succeeds and returns: ()

Exception:
invalid input

Extending XQuery with pattern matching: Basic pattern syntax

- A `text node` matches another `text node` (with "matching" string value)
- An `attribute="value"` matches another `attribute="value"` (with same name and "matching value")
- An `<element/> E` matches another `<element/> F` (with same name) that has
 - matching attributes (every attribute of E matches an attribute of F)
 - matching descendants (every *child* of E matches an *descendant* of F, such that the order of matched descendants of F is the same as the order of children of E)

Example

A simple pattern:

```
<element foo="bar">
  cat<meow/>
</element>
```

matches

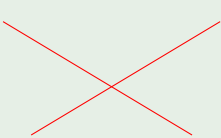
```
<element foo="bar">
  cat<meow/>
</element>
```

or

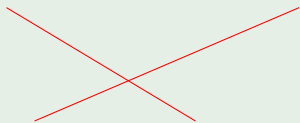
```
<element foo="bar" att="value">
  <p>cat<call><meow loudness="60 dB"/></call></p>
</element>
```

But it does **not** match (missing attribute)

nor (invalid order)



```
<element>
  cat<meow/>
</element>
```



```
<element foo="bar">
  <meow/>cat
</element>
```

If you have some ideas/suggestions about XML/HTML matching (e.g. direct, n-th child matching), just add them here:

-
-
-

Extending XQuery with pattern matching: Selecting data

We allow embedding of arbitrary XQuery expressions.

Example

```
<a><b>{.}</b></a>
```

matched against

```
<a><b>some text</b></a>
```

returns `.` evaluated with the context item `some text`

```
= <b>some text</b>
```

Example

```
<html> <h2>A</h2> <a{@href}</a> <h2>B</h2> </html>
```

matched against

```
<html>
  <a href="http://wrong">link</a>
  <h2>A</h2>
  <a href="http://example.org">link</a>
  <h2>B</h2>
  <a href="http://wrong">link</a>
</html>
```

returns

```
href="http://example.org"
```

(Alternative syntax: replace `{.}` with `<t:s>.</t:s>` or `<template:s>.</template:s>`)

How to return multiple values?

Allow variable assignments as named return values.

Example

```
<a> { $target := data(@href), $caption := text() } </a>
```

matched against

```
<a href="http://example.org">an example</a>
```

returns `$target = http://example.org`
and `$caption = an example`

Possible abbreviations:

```
<element>{$variable}</element>
⇒ <element>{$variable := .}</element>
```

```
<element attribute="{ $variable }"/>
⇒ <element>{$variable := @attribute}</element>
```

(actually `<element>{ @attribute / ($variable := .)}</element>`)

Advanced pattern syntax

Regular expressions have more operators: `?`, `+`, `*`, `|`, `[]`

What do our patterns have?

Optional elements:

`<element/?>`

Are ignored in the pattern, if they do not exist in the data

(Alternative syntax: add attribute `t:optional="true"` or `template:optional="true"`)

Repeated elements:

`<element/>+ or <element/>*`

Are repeated as long as possible

Example

`<a>{data(@href)}+`

matched against

`<html> 1 2 </html>`

returns `("#1", "#2")`

(Alternative syntax: surround with `<t:loop min="" max="">...</t:loop>` or `<template:loop>`)

Alternative elements

Multiple elements are accepted

Example

```
<t:switch>
  <a>{text()}</a>
  <b>{text()}</b>
</t:switch>
```

matched against either `<a>foobar` or `foobar`

returns `foobar`

(**Not an** alternative syntax: `(..|..)`. Should it be one?)

Many more:

`<t:loop><t:switch>` Used together \Rightarrow unordered matching

`<t:switch prioritized="true">` use element order of pattern instead order of data to find the “first” match

`t:condition=` Arbitrary XQuery condition for testing matching

`<t:if>` skip/use parts of the pattern depending on conditions

`<t:else>` opposite of `<t:if>`

`t:test=` abbreviation for `<t:if>`

`<t:meta>` change various matching options, e.g. exact match vs. regex match for text nodes

JSONiq (proposal)

Pattern matching is also useful for JSONiq types:

- atomar values match other atomar values that have the same value
- object O matches object P, if for every property of O there is a property of P with the same name and matching value
e.g. `{"a": 1}` matches `{"a": 1, "b": 2}`
- array A matches array B, if there is a subsequence of B with the same length as A, where the i-th member of A matches the i-th member of the subsequence
e.g. `[1,2,3]` matches `[1,2,"xxxx",3,4,5]`

Selector expressions could be contained in `<t:s>` nodes or dynamic functions

Example

```
      {"a":  [1,2,3], "b":  null, "c":  <t:s>.</t:s>}
```

applied to

```
      {"a":  [1,"u",2,"v",3], "b":  null, "c":  [7,8,9], "d":  17}
```

would return

```
      [7,8,9]
```

If you have some ideas about JSONiq, just add them here:

-
-
-
-

Integration in XQuery

A function for pattern matching

Every pattern is an XML/JSONiq element, so it can be stored in a variable. So we can make a function:

```
pxp:match($pattern as item(), $data as item(*)
```

Returning a map of all variables of the pattern, combining multiple assignments to a single sequence

A function for pattern matching

Example

```
pxp:match(<ul> <li>{{$var := text()}}</li>+ </ul>,  
          <ul> <li>1</li> <li>2</li> </ul>)
```

returns a map

```
{"var": ("1", "2") }
```

But calling a function in every query is inconvenient.

So we add the patterns to the `let`, `typeswitch` and `for` expression:

Example

```
let <html><h2>section 1</h2>  
    <p>{$var}</p>+  
    <h2>section 2</h2></html>  
:= <html><h2>section 1</h2>  
    <p>a</p>  
    <p>b</p>  
    <h2>section 2</h2>  
    <p>c</p></html>  
return $var / string()
```

would return

```
("a", "b")
```

Extending typeswitch

The `typeswitch` extensions is like the `let` extensions, but multiple patterns can be given in case clauses:

Example

```
typeswitch (<html><b>foobar</b></html>)  
case <a>{.}</a> return "a link to " || @href  
case <b>{$v}</b> return "bold text: " || $v  
default return "unknown element"
```

would return

```
"bold text: foobar"
```

Extending for

The `for` extension iterates over all assignments in that merging them:

Example

```
for <ul> <li>{.}</li>+ </ul>  
in <ul> <li>1</li> <li>2</li> </ul>  
return "li: " || .
```

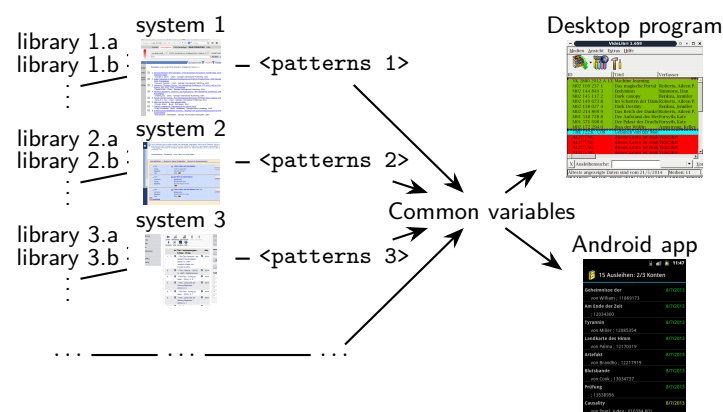
would return

```
("li: 1", "li: 2")
```

Available in our XQuery engine: <http://xidel.sourceforge.net>

Practical application

We have created an app/wrapper for 175 libraries by creating patterns for 15 different web catalogues/OPACs that let you search in the catalogues, see your lend books, renew them, automatically renew them before they are due.



(If your library wants to be in the app, too, I am happy to add them. Just ask (Benito van der Zander, benito@benibela.de). (Although so far it is entirely on German)

Creating a pattern:

It is very easy

Take a webpage:

[1/7]	<input type="checkbox"/>	Titel	:		<<The>> Dresden files / 1. Storm front (Bd. : 1)
		Verfasser	:		Butcher, Jim
		Verlag	:		Orbit
		Signatur	:		ENGL BUT 12/1:1
		Jahr	:		2011
[2/7]	<input type="checkbox"/>	Titel	:		<<The>> Dresden files / 6. Blood Rites (Bd. : 6)
		Verfasser	:		Butcher, Jim
		Signatur	:		ENGL BUT 12/1:6
		Jahr Auflage	:		2009 - Auflage : 18
[3/7]	<input type="checkbox"/>	Titel	:		<<The>> Dresden files / 5. Death masks (Bd. : 5)
		Verfasser	:		Butcher, Jim
		Signatur	:		ENGL BUT 12/1:5
		Jahr Auflage	:		2003 - Auflage : 1
[4/7]	<input type="checkbox"/>	Titel	:		<<The>> Dresden files / 4. Summer knight (Bd. : 4)
		Verfasser	:		Butcher, Jim

Take a webpage's source:

```
...
<table border="0" cellspacing="1" cellpadding="1" width="100%">
  <tr valign="top">
    <td width="90" nowrap align="left">
      Titel
    </td>
    <td width="10"></td>
  </tr>
  <tr>
    <td width="16"></td>
    <td width="16">
      <img align="middle" title="Buch" alt="book icon" />
    </td>
    <td>
      <a class="darkLink" href="APS_CAT_IDENTIFY? Style=Portal2&SubStyle=&Lang=GER&R...
      >
        &lt;The&gt; Dresden files / 1. Storm front
        (
          Bd. :&nbsp;1
        )
      </a>
    </td>
  </tr>
  <tr valign="top">
    <td width="90" nowrap align="left">
      Verfasser
    </td>
    <td width="10"></td>
    <td></td>
    <td></td>
  </tr>
  <tr>
    <td colspan="4">
      <span class="darkLink">
        Butcher , Jim
      </span>
    </td>
  </tr>
</table>
<tr valign="top">
```

Remove the nonsense (formatting, whitespace):

```
...
<table>
<tr>
<td>Titel</td><td>:</td><td></td><td><img alt="book icon" /></td>
<td><a>&lt;The&gt; Dresden files / 1. Storm front (Bd. :&nbsp;1)</a></td></tr>
<tr>
<td>Verfasser</td><td>:</td><td></td><td></td>
<td><span>Butcher , Jim</span></td></tr>
<tr>
<td>Verlag</td><td>:</td><td></td><td></td>
<td>Orbit</td></tr>
<tr>
<td><b>Signatur</b></td><td>:</td><td></td><td></td>
<td>ENGL BUT 12/1:1</td></tr>
<tr>
<td><b>Jahr</b></td><td>:</td><td></td><td></td>
<td>2011</td></tr>
</table>
..
```

⇒ Finished pattern

Rarely takes more than a few minutes for a page (although it does not always then work then, e.g. due to javascript, invalid html, ...)