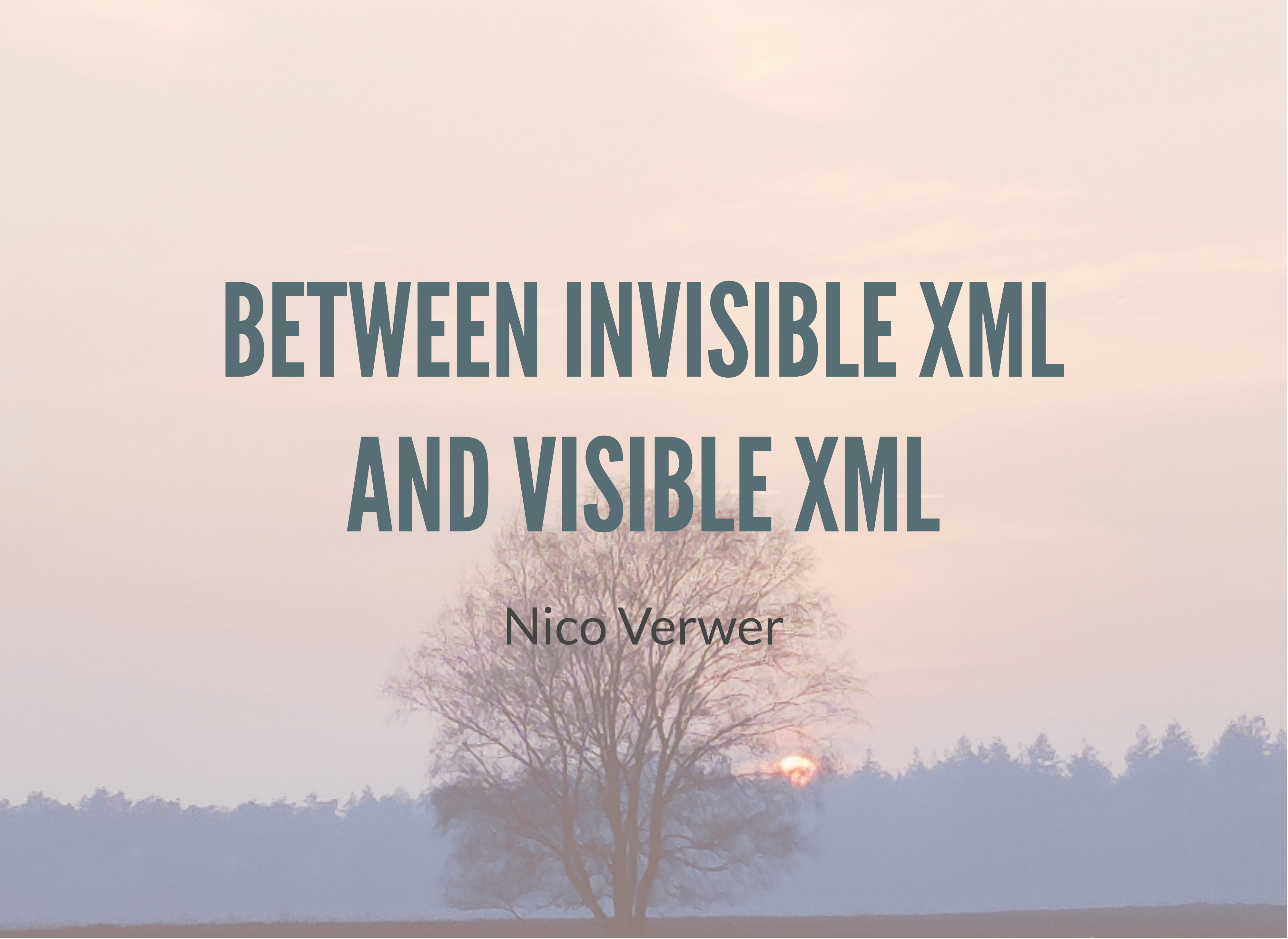
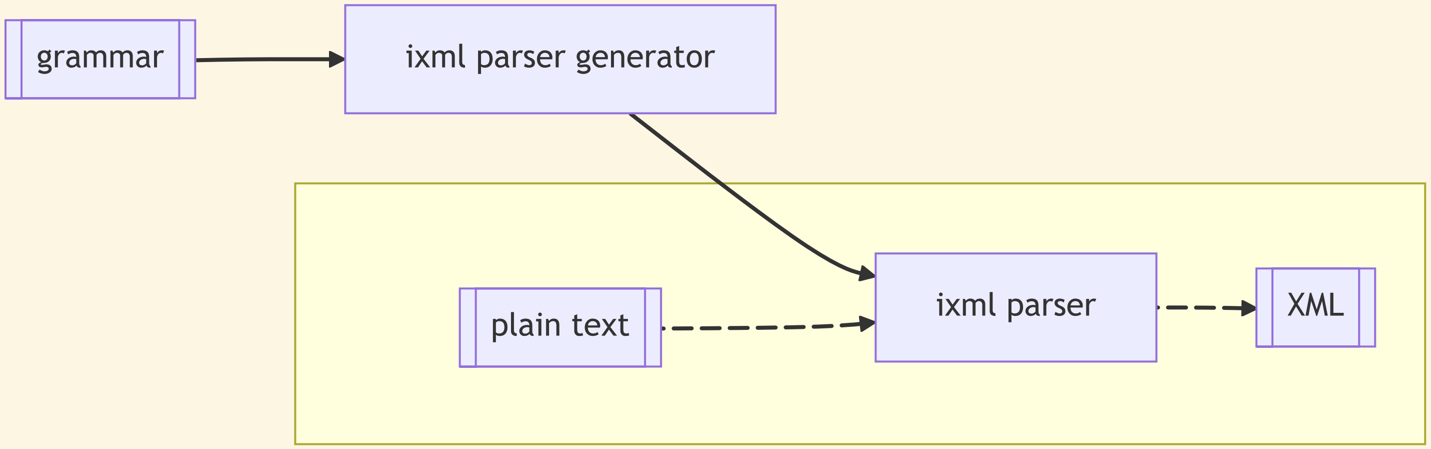


BETWEEN INVISIBLE XML AND VISIBLE XML

Nico Verwer

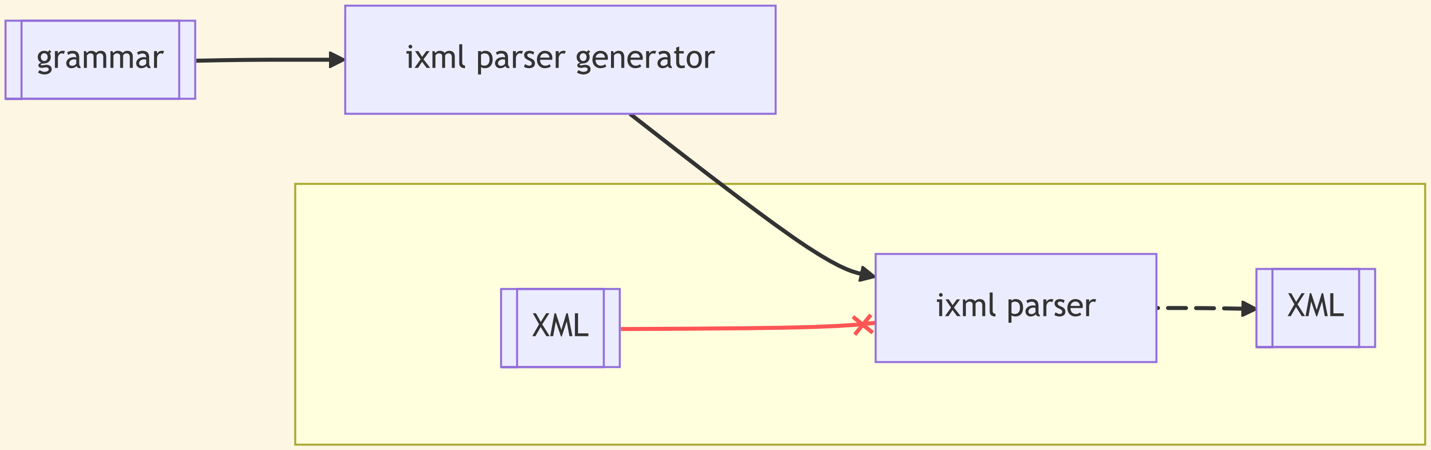
ixml parsers parse *text*



real documents are XML ☺

Text fragments in the visible XML can contain invisible XML.

parsing → more structure → enriched documents



what kind of 'real' documents?

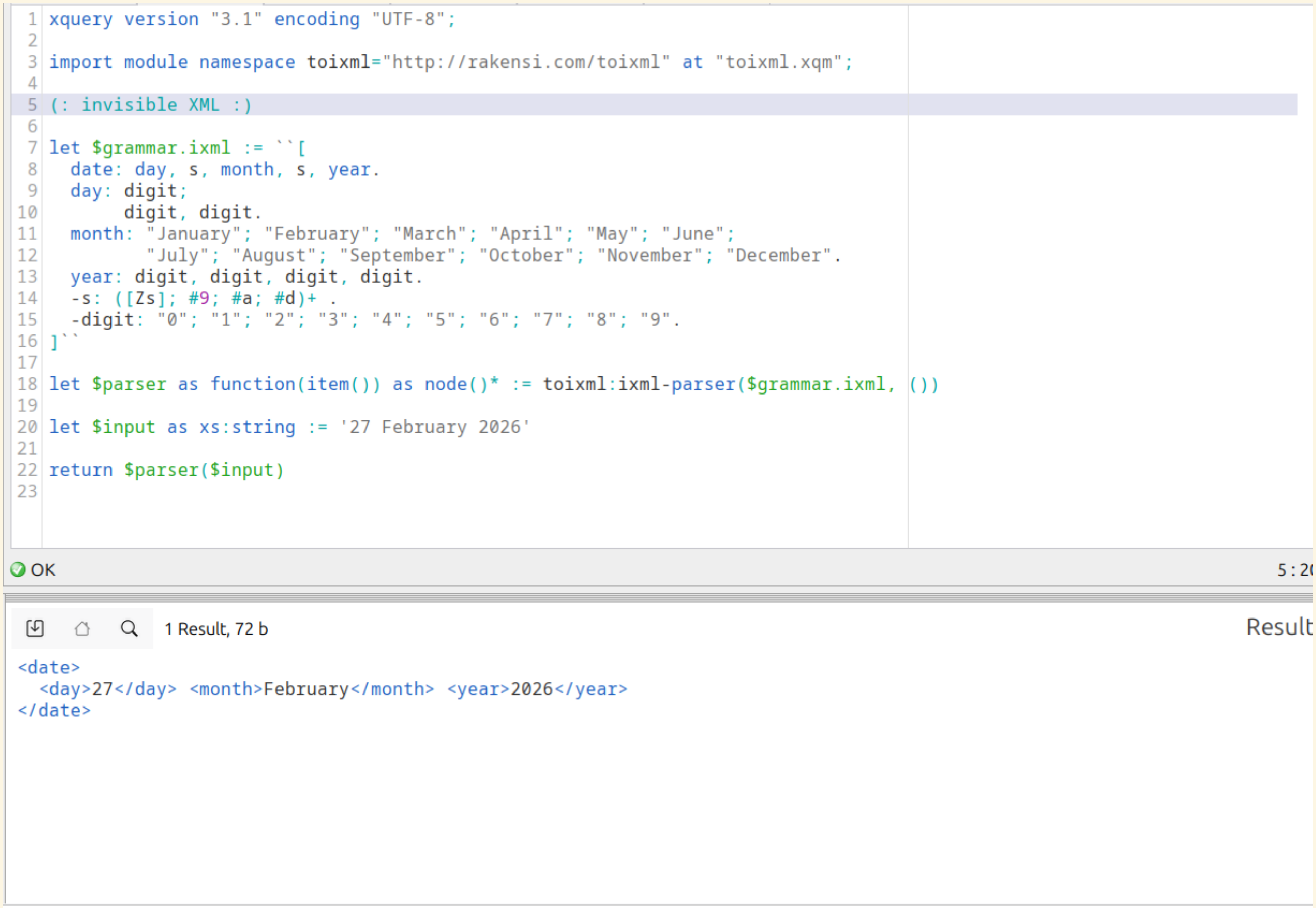
> *“ The 'Link eXtractor' (LX) recognizes references to legal sources, like legislation, court decisions, parliamentary documents and official publications in textual documents [...] by detecting fixed strings and rule-based patterns. ”*

— https://gitlab.com/koop/ld/lx/lx-core

invisible XML

'hello world'

```
date: day, s, month, s, year.
day: digit;
    digit, digit.
month: "January"; "February"; "March"; "April"; "May";
      "June"; "July"; "August"; "September"; "October";
      "November"; "December".
year: digit, digit, digit, digit.
-s: ([Zs]; #9; #a; #d)+ .
-digit: "0"; "1"; "2"; "3"; "4"; "5"; "6"; "7"; "8"; "9".
```



```
xquery version "3.1" encoding "UTF-8";

import module namespace toixml="http://rakensi.com/toixml" at "toixml.xqm";

(: invisible XML :)

let $grammar.ixml := ``[
  date: day, s, month, s, year.
  day: digit;
       digit, digit.
  month: "January"; "February"; "March"; "April"; "May"; "June";
         "July"; "August"; "September"; "October"; "November"; "December".
  year: digit, digit, digit, digit.
  -s: ([Zs]; #9; #a; #d)+ .
  -digit: "0"; "1"; "2"; "3"; "4"; "5"; "6"; "7"; "8"; "9".
]``

let $parser as function(item()) as node()* := toixml:ixml-parser($grammar.ixml, ())

let $input as xs:string := '27 February 2026'

return $parser($input)
```

```
<date>
  <day>27</day> <month>February</month> <year>2026</year>
</date>
```

transparent XML

Suppose we have already recognized months.

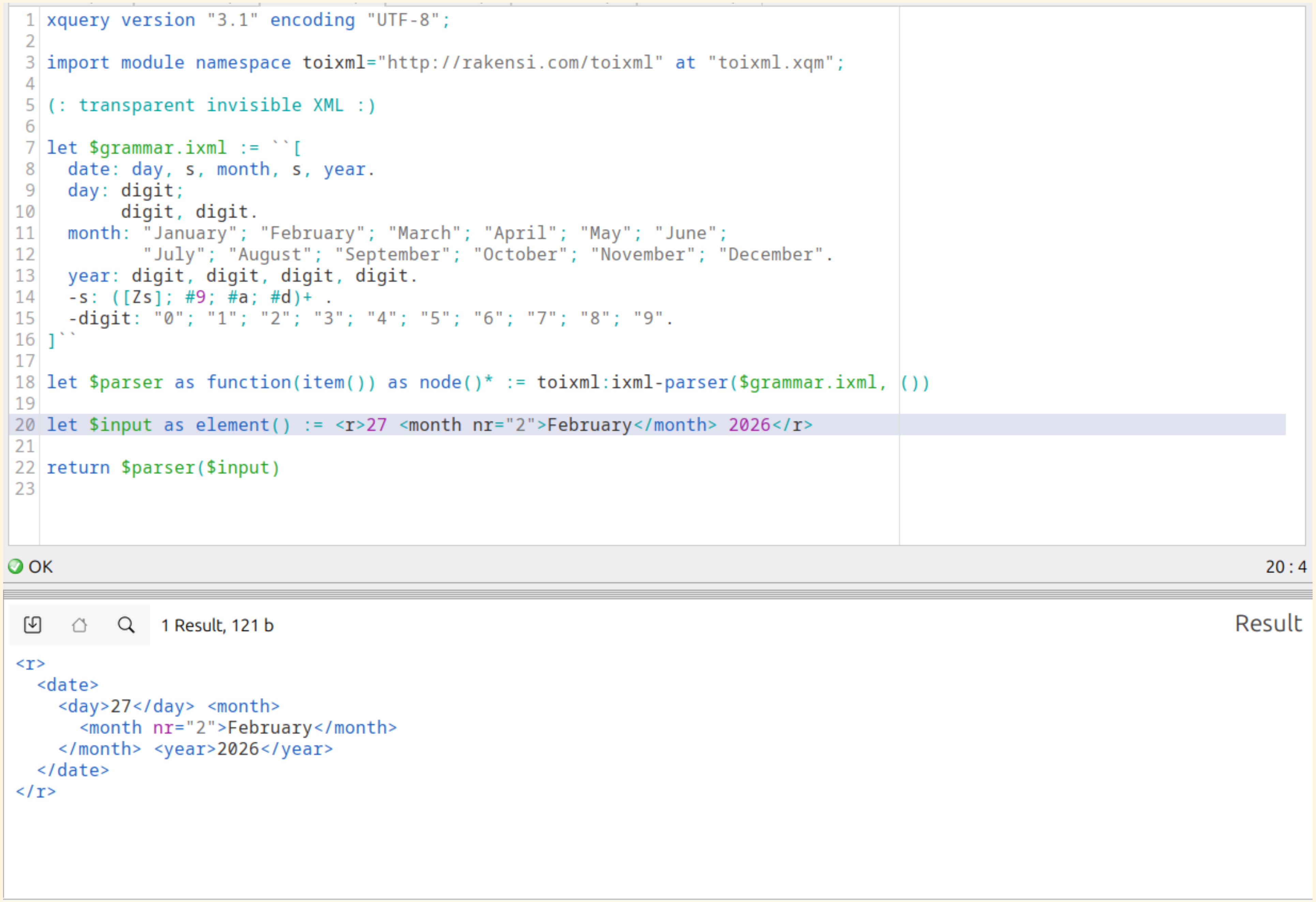
```
<r>27 <month nr="2">February</month> 2026</r>
```

Keep the markup by making it *transparent*.

```
<r><date>
  <day>27</day>
  <month><month nr="2">February</month></month>
  <year>2026</year>
</date></r>
```

two `<month>` elements ☹

parse `<month>` again ☹



```
xquery version "3.1" encoding "UTF-8";

import module namespace toixml="http://rakensi.com/toixml" at "toixml.xqm";

(: transparent invisible XML :)

let $grammar.ixml := ``[
  date: day, s, month, s, year.
  day: digit;
       digit, digit.
  month: "January"; "February"; "March"; "April"; "May"; "June";
         "July"; "August"; "September"; "October"; "November"; "December".
  year: digit, digit, digit, digit.
  -s: ([Zs]; #9; #a; #d)+ .
  -digit: "0"; "1"; "2"; "3"; "4"; "5"; "6"; "7"; "8"; "9".
]``

let $parser as function(item()) as node()* := toixml:ixml-parser($grammar.ixml, ())

let $input as element() := <r>27 <month nr="2">February</month> 2026</r>

return $parser($input)
```

OK 20 : 4

1 Result, 121 b

Result

```
<r>
  <date>
    <day>27</day> <month>
      <month nr="2">February</month>
    </month> <year>2026</year>
  </date>
</r>
```

opaque XML

Use the `<month>` element that is already there by making it *opaque*.

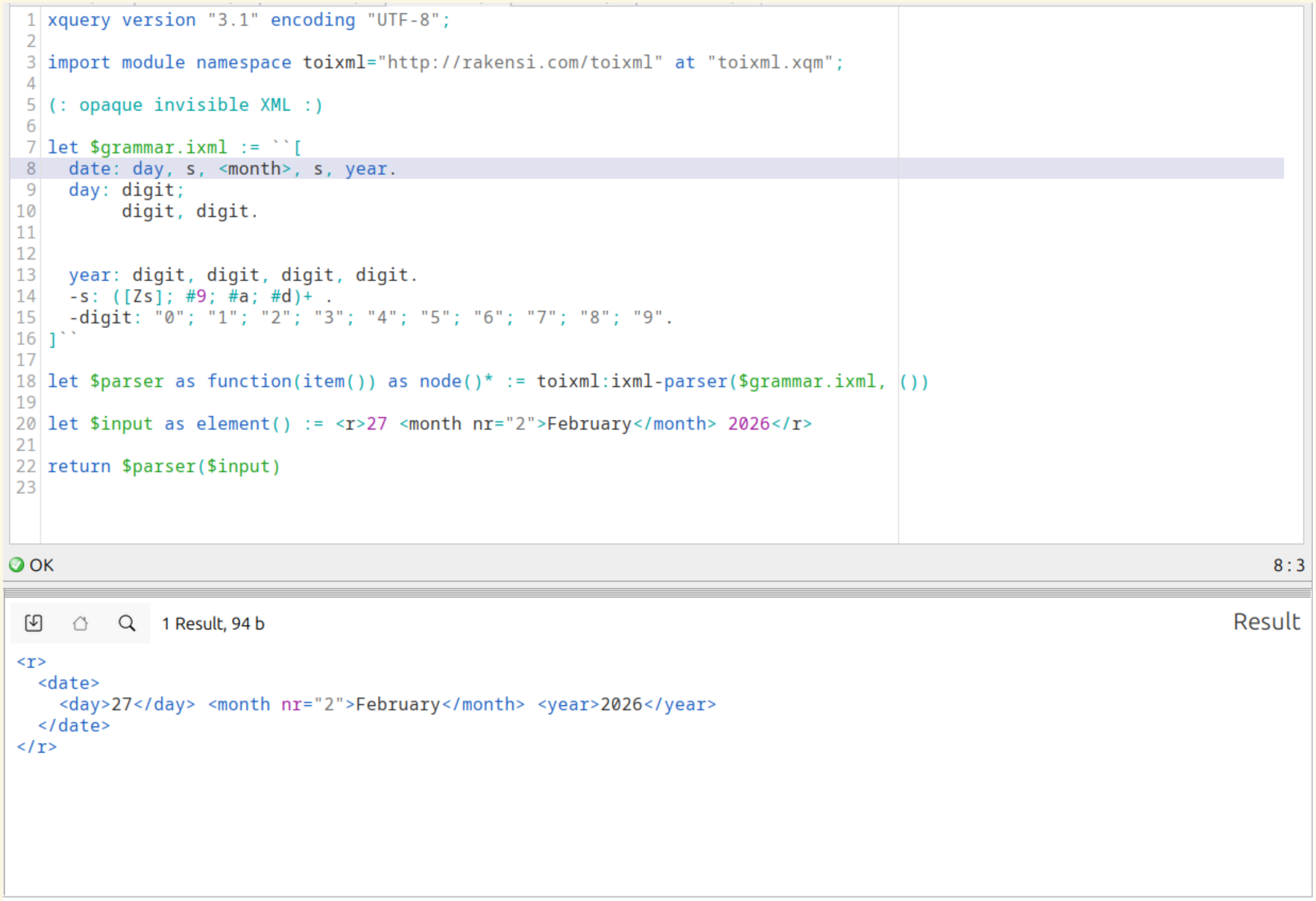
tokenization, multi-stage parsing pipelines

```
date: day, s, <month>, s, year.
```

`<month>` is a 'pre-parsed non-terminal'

The parser checks if a `<month>` element is present at the right position.

One day, we might also match attributes.



```
xquery version "3.1" encoding "UTF-8";

import module namespace toixml="http://rakensi.com/toixml" at "toixml.xqm";

(: opaque invisible XML :)

let $grammar.ixml := ``[
  date: day, s, <month>, s, year.
  day: digit;
       digit, digit.


  year: digit, digit, digit, digit.
  -s: ([Zs]; #9; #a; #d)+ .
  -digit: "0"; "1"; "2"; "3"; "4"; "5"; "6"; "7"; "8"; "9".
]``

let $parser as function(item()) as node()* := toixml:ixml-parser($grammar.ixml, ())

let $input as element() := <r>27 <month nr="2">February</month> 2026</r>

return $parser($input)
```

OK 8 : 3

1 Result, 94 b

Result

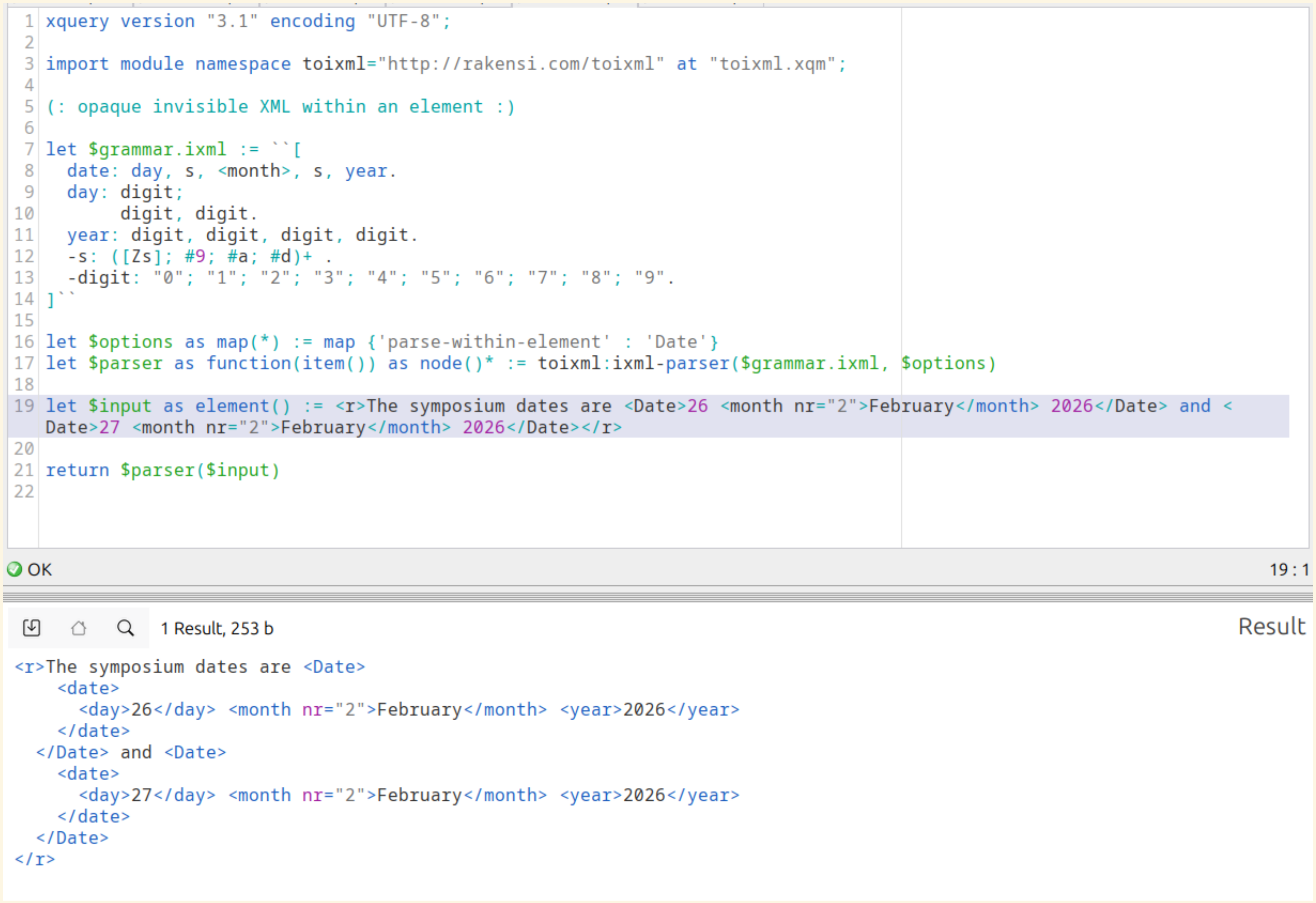
```
<r>
  <date>
    <day>27</day> <month nr="2">February</month> <year>2026</year>
  </date>
</r>
```

parsing *within* elements

```
<r>The symposium dates are
  <Date>26 <month nr="2">February</month> 2026</Date>
  and
  <Date>27 <month nr="2">February</month> 2026</Date>
</r>
```

```
let $options as map(*) := map {'parse-within-element' : 'Date'}
```

```
<r>The symposium dates are <Date>
    <date>
      <day>26</day> <month nr="2">February</month> <year>2026</ye
    </date>
  </Date> and <Date>
    <date>
      <day>27</day> <month nr="2">February</month> <year>2026</ye
    </date>
  </Date>
</r>
```



```
xquery version "3.1" encoding "UTF-8";

import module namespace toixml="http://rakensi.com/toixml" at "toixml.xqm";

(: opaque invisible XML within an element :)

let $grammar.ixml := ``[
  date: day, s, <month>, s, year.
  day: digit;
       digit, digit.
  year: digit, digit, digit, digit.
  -s: ([Zs]; #9; #a; #d)+ .
  -digit: "0"; "1"; "2"; "3"; "4"; "5"; "6"; "7"; "8"; "9".
]``

let $options as map(*) := map {'parse-within-element' : 'Date'}
let $parser as function(item()) as node()* := toixml:ixml-parser($grammar.ixml, $options)

let $input as element() := <r>The symposium dates are <Date>26 <month nr="2">February</month> 2026</Date> and <
Date>27 <month nr="2">February</month> 2026</Date></r>

return $parser($input)
```

OK 19 : 1

1 Result, 253 b

Result

```
<r>The symposium dates are <Date>
    <date>
      <day>26</day> <month nr="2">February</month> <year>2026</year>
    </date>
  </Date> and <Date>
    <date>
      <day>27</day> <month nr="2">February</month> <year>2026</year>
    </date>
  </Date>
</r>
```

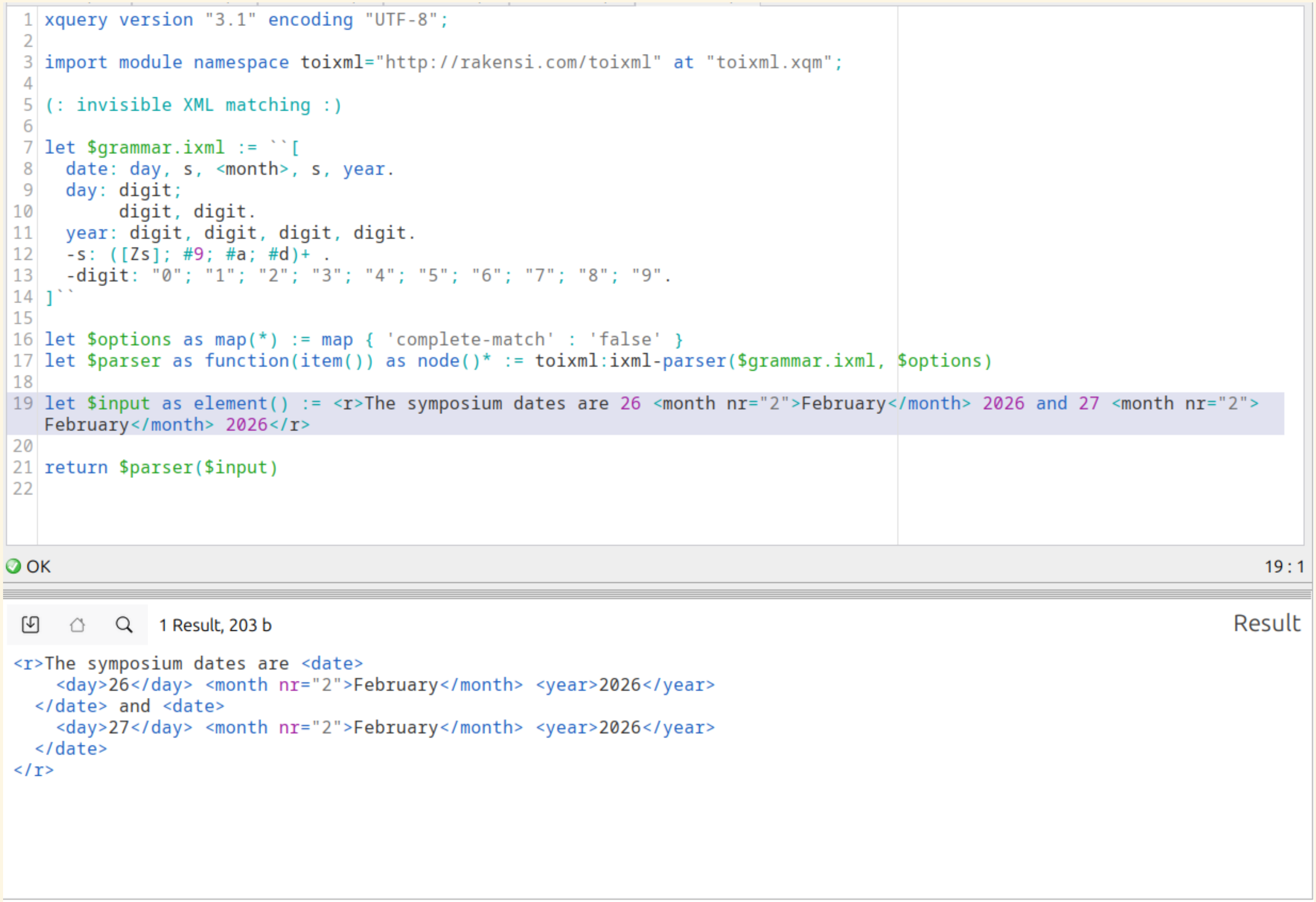
invisible XML *matching*

```
<r>The symposium dates are
  26 <month nr="2">February</month> 2026 and
  27 <month nr="2">February</month> 2026
</r>
```

```
let $options as map(*) := map { 'complete-match' : 'false' }
```

```
<r>The symposium dates are <date>
    <day>26</day> <month nr="2">February</month> <year>2026</year
  </date> and <date>
    <day>27</day> <month nr="2">February</month> <year>2026</year
  </date>
</r>
```

This will find the longest matching fragments.



```
xquery version "3.1" encoding "UTF-8";

import module namespace toixml="http://rakensi.com/toixml" at "toixml.xqm";

(: invisible XML matching :)

let $grammar.ixml := ``[
  date: day, s, <month>, s, year.
  day: digit;
       digit, digit.
  year: digit, digit, digit, digit.
  -s: ([Zs]; #9; #a; #d)+ .
  -digit: "0"; "1"; "2"; "3"; "4"; "5"; "6"; "7"; "8"; "9".
]``

let $options as map(*) := map { 'complete-match' : 'false' }
let $parser as function(item()) as node()* := toixml:ixml-parser($grammar.ixml, $options)

let $input as element() := <r>The symposium dates are 26 <month nr="2">February</month> 2026 and 27 <month nr="2">
February</month> 2026</r>

return $parser($input)
```

OK 19 : 1

1 Result, 203 b Result

```
<r>The symposium dates are <date>
    <day>26</day> <month nr="2">February</month> <year>2026</year>
  </date> and <date>
    <day>27</day> <month nr="2">February</month> <year>2026</year>
  </date>
</r>
```

implementation

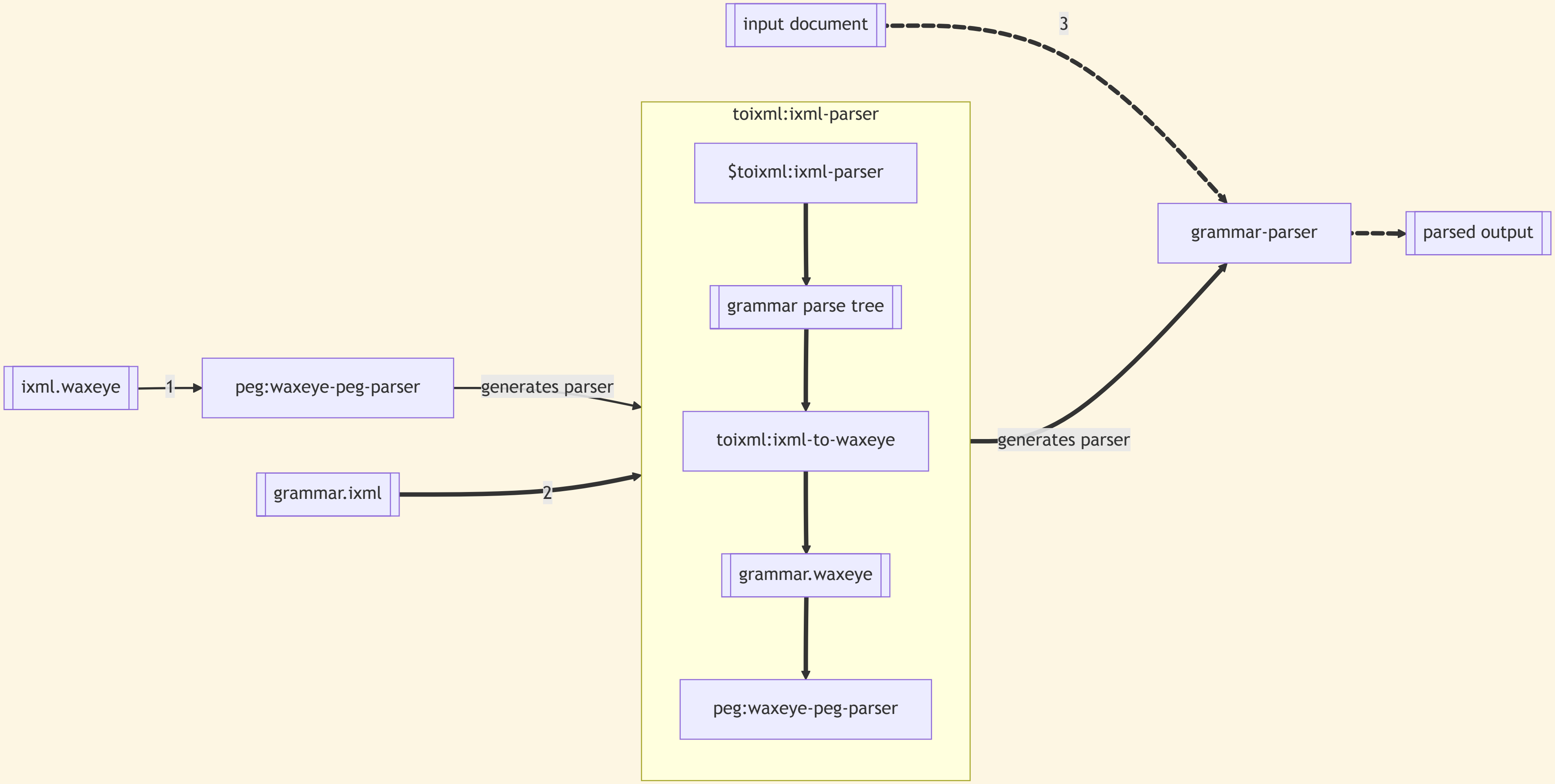
LX uses a PEG parser generator (waxeye), not ixml.

Working on an ixml - PEG implementation,
which will be open source.

The waxeye implementation is not open source.

The examples use a *transpiler*.

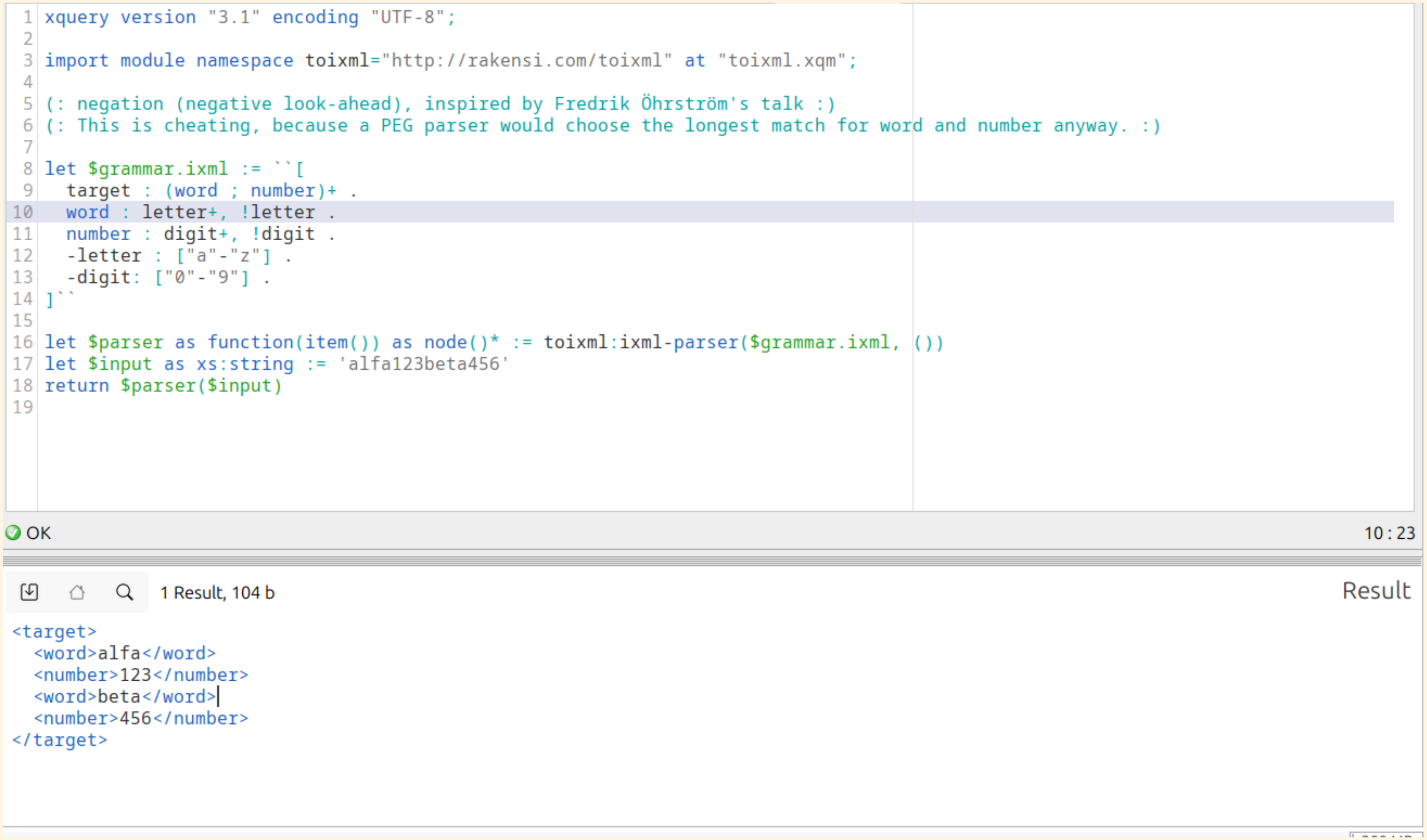
not compliant (@, -, +, character classes)


input document
3
toixml:ixml-parser
$toixml:ixml-parser
grammar-parser
parsed output
grammar parse tree
ixml.waxeye
1
peg:waxeye-peg-parser
generates parser
toixml:ixml-to-waxeye
generates parser
grammar.ixml
2
grammar.waxeye
peg:waxeye-peg-parser

if there is time left

negation (Fredrik Öhrström's talk)

partial binding / parsing pipeline (John Lumley's talk)



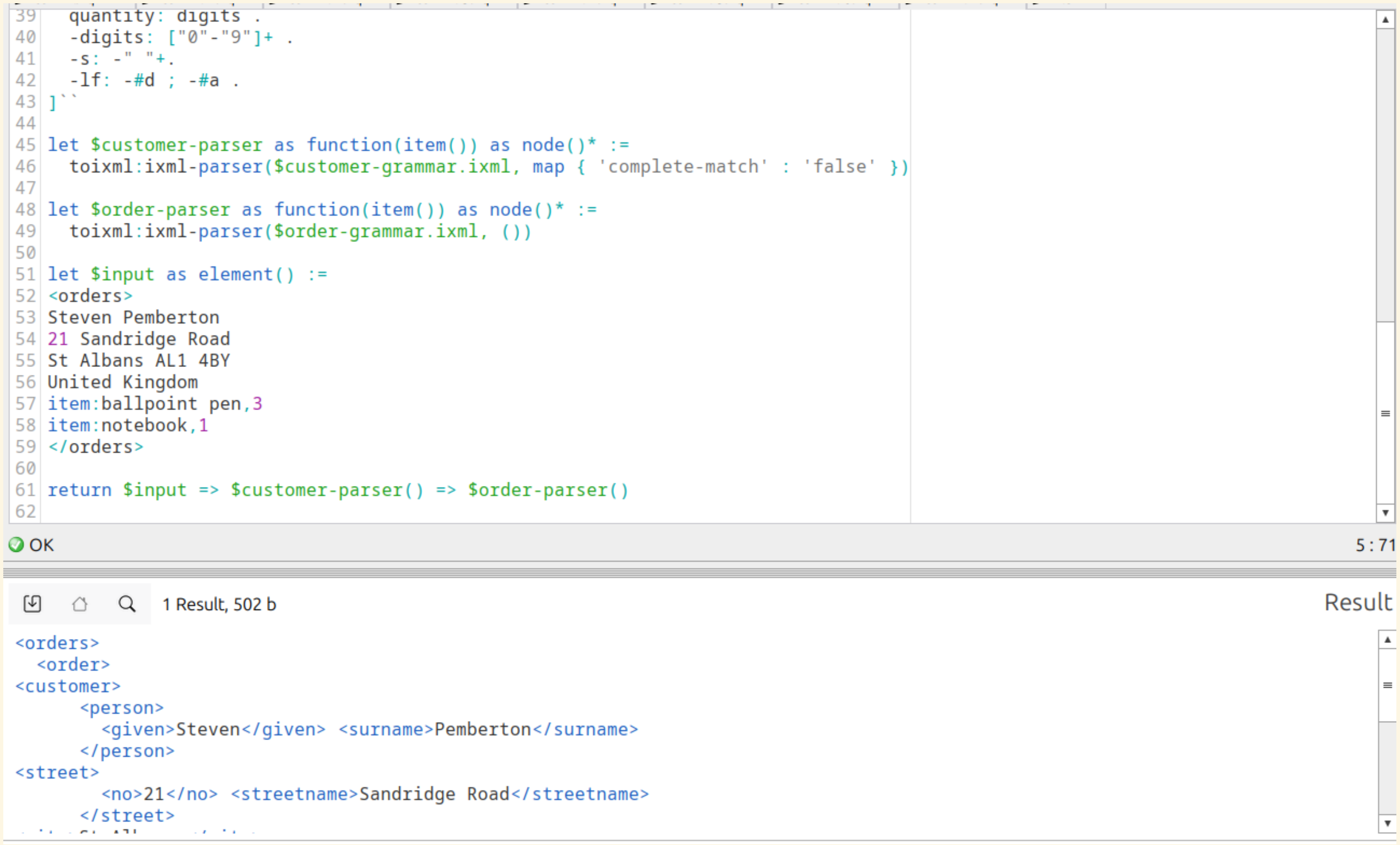
```
xquery version "3.1" encoding "UTF-8";

import module namespace toixml="http://rakensi.com/toixml" at "toixml.xqm";

(: negation (negative look-ahead), inspired by Fredrik Öhrström's talk :)
(: This is cheating, because a PEG parser would choose the longest match for word and number anyway. :)

let $grammar.ixml := ``[
  target : (word ; number)+ .
  word : letter+, !letter .
  number : digit+, !digit .
  -letter : ["a"-"z"] .
  -digit: ["0"-"9"] .
]``

let $parser as function(item()) as node()* := toixml:ixml-parser($grammar.ixml, ())
let $input as xs:string := 'alfa123beta456'
return $parser($input)
```

OK 10 : 23

1 Result, 104 b

Result

```
<target>
  <word>alfa</word>
  <number>123</number>
  <word>beta</word>
  <number>456</number>
</target>
```



```
  quantity: digits .
  -digits: ["0"-"9"]+ .
  -s: -" "+.
  -lf: -#d ; -#a .
]``

let $customer-parser as function(item()) as node()* :=
  toixml:ixml-parser($customer-grammar.ixml, map { 'complete-match' : 'false' })

let $order-parser as function(item()) as node()* :=
  toixml:ixml-parser($order-grammar.ixml, ())

let $input as element() :=
<orders>
Steven Pemberton
21 Sandridge Road
St Albans AL1 4BY
United Kingdom
item:ballpoint pen,3
item:notebook,1
</orders>

return $input => $customer-parser() => $order-parser()
```

OK 5 : 71

1 Result, 502 b Result

```
<orders>
  <order>
<customer>
      <person>
        <given>Steven</given> <surname>Pemberton</surname>
      </person>
<street>
        <no>21</no> <streetname>Sandridge Road</streetname>
      </street>
```

LX's parsing pipeline (small fragment)

```
( $input
=> (: named entity recognition for legislation / regulations :)
  transform:if( $extract-leg-reg,
    lx-ner:regeling(?, $is-test, $is-recognize)
  )
=> (: parse references to EU legislation / regulations :)
  transform:if( $extract-eu-leg-reg,
    ( lx-peg:eu-regelgeving(?)
    , transform:xsl('links/xsl/normalize-eu-regelgeving.xslt', ())
    , lido:resolve-celex(?, $options?test-url)
    )
  )
=> (: local alias detection :)
  transform:if( $extract-leg-reg,
    ( lx-peg:lokale-alias(?)
```

conclusion

invisible XML: parse text input into XML output

visible XML: parse text from XML input

transparent XML: ignore and keep XML structure

opaque XML: pre-parsed non-terminals

parse within specific elements

match invisible XML fragments