A Declarative Code Browser with ixml and XForms

Steven Pemberton, CWI, Amsterdam

2 **Contents**

- Contents
- Abstract
- ixml
- Small ixml example
- Process
- Process
- Process
- ixampl: the pilot implementation
- Structure of the implementation
- ABC
- Characters
- Digression: Unicode and UTF-8
- Digression digression
- Digression digression
- Digression digression
- Digression digression
- Digression digression
- UTF-8
- UTF-8: The traditional view
- The problem for ABC
- UTF-8 in ixml
- Solution
- Use of ixml
- Result

4 **Abstract**

ixml: a declarative language for transforming data representations. stable specification: 2022.

The original pilot implementation of ixml, ixampl, was written in the very-high-level programming language, ABC, the forerunner of Python.

The implementation is split into two parts: a bootstrap parser, that reads ixml grammars and transforms them into the structure needed for part 2, which is a generic parser that reads any document and transforms it into XML.

Part 1 is about 700 lines of code, part two about 780.

One of the many possible uses of ixml is to transform any data representation into XML so that it can be used as input to XForms, a Turing-complete declarative programming language that uses XML as its data format.

To illustrate this, a code browser was made for the ixampl implementation, in a nicely self-referential way, using a 30 line ixml grammar to transform the ABC code into an XML representation, and use this as input to an XForms application of around 120 lines that enables you to browse and search in the ixampl code.

Although the browser is for ABC code, with the exception of the ixml grammar there is little that is specific to ABC, meaning that it would be easy to adapt it for another language.

So combining two declarative technologies, we were able to create a useful, functional code browser in only 150 lines of code.

Keywords: ixml, declarative languages, declarative programming, ABC, XForms, UTF-8

5 **ixml**

ixml: a declarative language for transforming textual documents with implied structure into documents where the structure has been made explicit.

Stable specification: 2022.

"Community group" at W3C.

6 **Small ixml example**

Input:

```
3 November 2023
```

Data description:

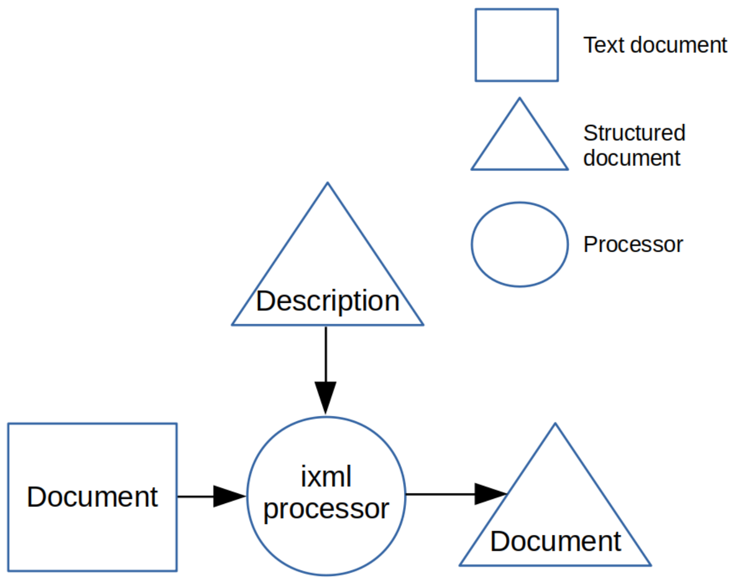
```
date: day, -" ", month, -" ", year.
day: d, d?.
month: "January"; ...; "December".
year: d, d, d, d.
-d: ["0".."9"].
```

Output:

```
<date>
  <day>3</day>
  <month>November</month>
  <year>2023</year>
</date>
```

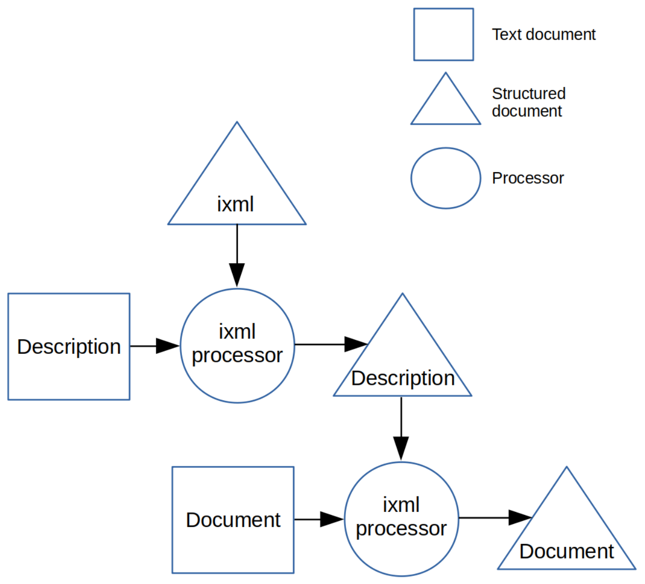
7 Process

Transform a textual document into a structured document



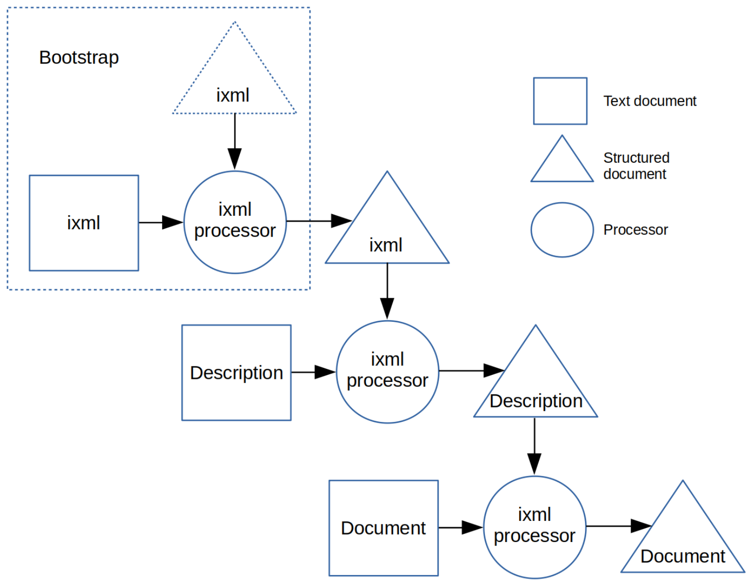
8 Process

The description of the format is also a flat textual document, so that too is transformed



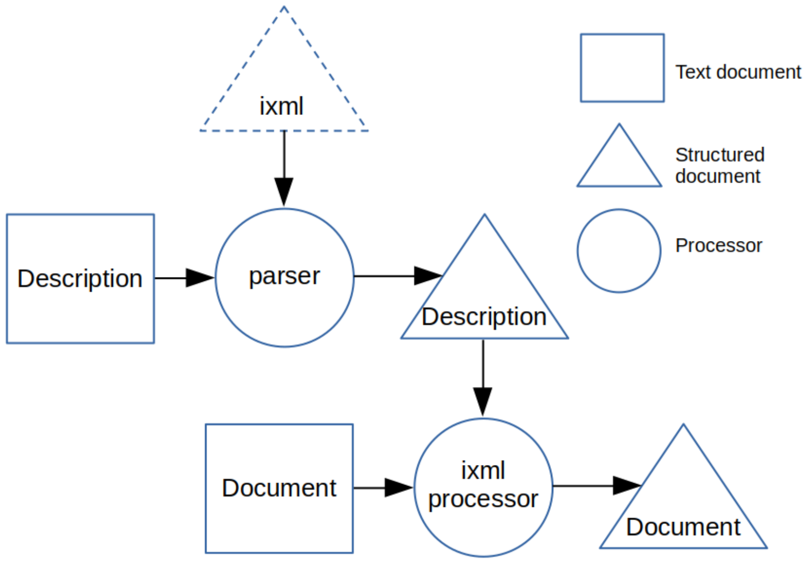
9 **Process**

ixml is described in ixml as well, so the structured version of that is produced in a bootstrap phase



10 **ixampl: the pilot implementation**

The pilot implementation however has the description of ixml hardwired



11 **Structure of the implementation**

So the implementation consists of two parts:

Part 1, for reading and parsing the ixml description, and serialising into the format expected by part two.

Part 2, for using that description to parse the text document, and serialise it as XML

12 **ABC**

The implementation is written in the very-high-level interactive language ABC.

It has only 5 data types: Numbers, Strings, Compounds, Bags, Maps. Bags and maps are kept sorted.

All data (even numbers) is unbounded, and dynamic.

Statically typed without declarations.

It was the basis of Python.

13 **Characters**

ixml uses Unicode.

One problem with ABC is that it predates Unicode, and only has 8 bit characters.

Therefore no Unicode...

14 **Digression: Unicode and UTF-8**

Character sets are just a mapping from numbers (the 'code-point') to character representations.

The first computer I used had a 6-bit character set. 64 characters. No cased letters!

ASCII was a 7 bit character set giving 128 characters.

Latin1 then used the spare 8th bit to add another 128 characters.

And then came Unicode at 16 bits. Clearly, 65536 characters should be enough for anyone! (They really said that: *In a properly engineered design, 16 bits per character are more than sufficient for this purpose.)*

15 **Digression: Unicode and UTF-8**

Character sets are just a mapping from numbers (the 'code-point') to character representations.

The first computer I used had a 6-bit character set. 64 characters. No cased letters!

ASCII was a 7 bit character set giving 128 characters.

Latin1 then used the spare 8th bit to add another 128 characters.

And then came Unicode at 16 bits. Clearly, 65536 characters should be enough for anyone! (They really said that: *In a properly engineered design, 16 bits per character are more than sufficient for this purpose.)*

But it wasn't, and they went to, currently, 21 bits (10FFFF being the current highest character). 1,114,111 characters should be enough for anyone.

16 **Digression digression**

1114111: a lovely palindromic number!

17 **Digression digression**

1114111: a lovely palindromic number!

Whoever invented the term "palindrome" ought to be shot!

18 **Digression digression**

1114111: a lovely palindromic number!

Whoever invented the term "palindrome" ought to be shot!

Unfortunately, it was introduced by English poet and writer Henry Peacham in 1638. So he's already dead.

19 **Digression digression**

1114111: a lovely palindromic number!

Whoever invented the term "palindrome" ought to be shot!

Unfortunately, it was introduced by English poet and writer Henry Peacham in 1638.
So he's already dead.

My proposal:

20 **Digression digression**

1114111: a lovely palindromic number!

Whoever invented the term "palindrome" ought to be shot!

Unfortunately, it was introduced by English poet and writer Henry Peacham in 1638.
So he's already dead.

My proposal: wordrow

21 **UTF-8**

Up to Unicode, all characters fitted into one 8-bit byte.

21 bits don't fit in one byte, so there has to be a multi-byte encoding.

UTF-8 was born because of a problem with Unix-like systems: files had no metadata for character encodings: all files used the same encoding.

UTF-8 is a variable-length encoding of Unicode, with ASCII as a subset: so suddenly all existing Unix ASCII files were immediately Unicode-compliant!

22 **UTF-8: The traditional view**

UTF-8 is normally sold as a bit-oriented encoding: a character consists of 1-4 bytes, where the first few bits of each byte tell you the role of that byte.

Each byte either starts a character, or is a continuation byte.

"The first few bits" means: there are a number of 1 bits followed by a zero stop bit; all remaining bits are data. Count the number of leading 1 bits:

- 0: this is a single-byte ASCII character
- 1: this is a continuation byte
- 2: start of 2-bytes, 1 continuation byte follows
- 3: start of 3-bytes, 2 continuation bytes follow
- 4: start of 4-bytes, 3 continuation bytes follow.

Concatenate the data bits of the n bytes to give you the code-point.

23 **The problem for ABC**

ABC predates Unicode, and has 8 bit characters, but they are atomic: you can't look at the bits.

Aha moment: there are only 256 bytes, and each has only one role.

> 0-127: ASCII
> 128-191: continuation character
> 192-223: start a 2 byte character
> 224-239: start a 3 byte character
> 240-247: start a 4 byte character
> 248-255: illegal (5 or more leading 1 bits)

24 **The problem for ABC**

ABC predates Unicode, and has 8 bit characters, but they are atomic: you can't look at the bits.

Aha moment: there are only 256 bytes, and each has only one role.

> 0-127: ASCII
> 128-191: continuation character
> 192-223: start a 2 byte character
> 224-239: start a 3 byte character
> 240-247: start a 4 byte character
> 248-255: illegal (5 or more leading 1 bits)

(The illegal characters are available for a future expansion, allowing up to a 42 bit representation of characters, or in other words, 4,398,046,511,104 characters maximum, which should be enough for anyone!)

25 **UTF-8 in ixml**

Put another way:

```
input: u*.
u: u1; u2; u3; u4.

u1: b1.                      {ASCII,  7 bits}
u2: b2, b0.               {#80-#7FF, 11 bits}
u3: b3, b0, b0.         {#800-#FFFF, 16 bits}
u4: b4, b0, b0, b0. {#10000-#10FFFF, 21 bits}

b1: [#0-#7F].  {7 bits of data}
b0: [#80-#BF]. {adds 6 bits of data}
b2: [#C0-#DF]. {adds 5 bits of data}
b3: [#E0-#EF]. {adds 4 bits of data}
b4: [#F0-#F7]. {adds 3 bits of data}

{Continuation characters never start a character}
{#F8-#FF are illegal everywhere}
```

26 **Solution**

Record for each byte its role: how many bytes it starts.

1 for ASCII, 2 for the start of a 2 byte character, etc, 0 for continuation and illegal characters.

ABC has a `|` string operator:

```
"dishonest"|4 = "dish"
```

So `string|1` is the first byte, and `string|0` is an empty string.

And `string|start[string|1]` is the next Unicode character!

If it's an empty string, there was an encoding error.

All without bit-twiddling!

27 **Use of ixml**

Getting other things than XML into the XML pipeline.

E.g. biblio

```
[spec] Steven Pemberton (ed.), Invisible XML Specification,
 invisiblexml.org, 2022,
 https://invisiblexml.org/ixmlspecification.html
```

was processed by an ixml grammar whose top-level rules were something like

```
bibliography: biblioentry+.
biblioentry: abbrev, (author; editor), -", ",
   title, -", ", publisher, -", ",
   pubdate, -",",
   (artpagenums, -", ")?,
   (bibliomisc; biblioid)**-", ", -#a.
```

28 **Result**

```
<biblioentry>
   <abbrev>spec</abbrev>
   <editor>
      <personname>
         <firstname>Steven</firstname>
         <surname>Pemberton</surname>
      </personname>
   </editor>
   <title>Invisible XML Specification</title>
   <publisher>invisiblexml.org</publisher>
   <pubdate>2022</pubdate>
   <bibliomisc>
       <link xlink-href='https://invisiblexml.org/ixml-specification.html'/>
   </bibliomisc>
</biblioentry>
```

29 **Use of ixml**

Just as UTF-8 made all ASCII files Unicode, so ixml makes all text XML, only invisibly.

For example, as input to XForms.

Nicely self-referential application: make an XForms app for browsing the code of the ixml implementation itself.

30 **Code browser**

Convert the code to XML

Input that XML into an XForm for browsing

31 **Input**

ABC, like Python, uses indentation for grouping, and so is not context-free. However, it can still be parsed with ixml up to a maximum indentation.

```
\================== ixml system
\  Thu 31 Aug 15:44:15 CEST 2023
HOW TO IXML ixml WITH input ENDLF endlf:
      SHARE trace, g
      GRAMMAR ixml IN g AT root
      SELECT:
            g <> {}: \Parsed OK
               INIT INPUT input ENDLF endlf
               PARSE input WITH g AT root
               IF error.free:
                  SERIALISE root FROM trace WITH g USING input
            ELSE:
               WRITE "Failed" /
```

32 **ixml for ABC**

```
abc: (documentation?, how-to)+, documentation?.

documentation: caption, commentary?.
caption: -"\=", -"="*, ~[#a; "="], ~[#a]*, -#a+.
commentary: talk+.
-talk: -"\", ~[#a]*, #a+.

how-to: -"HOW TO ", header, comment?, -#a,
             body,
             refinement*,
             blank-lines.
header: ~[#a; "\"]*.
comment: -"\", ~[#a]*.
-body: line1+.
refinement: name, comment?, -#a, body.
name: [L], [L; " "]*, ":".
-blank-lines: -#a*.

line1: in,                                line, line2*.
line2: in, in,                            line, line3*.
line3: in, in, in,                        line, line4*.
line4: in, in, in, in,                    line, line5*.
line5: in, in, in, in, in,                line, line6*.
line6: in, in, in, in, in, in,            line, line7*.
line7: in, in, in, in, in, in, in,        line, line8*.
```

33 **Example Output**

```
<abc>
   <documentation>
      <caption> ixml system</caption>
      <commentary>  Thu 31 Aug 15:44:15 CEST 2023
</commentary>
   </documentation>
   <how-to>
      <header>IXML ixml WITH input ENDLF endlf:</header>
      <line1>SHARE trace, g</line1>
      <line1>GRAMMAR ixml IN g AT root</line1>
      <line1>SELECT:
         <line2>g &lt;> {}:
            <comment>Parsed OK</comment>
            <line3>INIT INPUT input ENDLF endlf</line3>
            <line3>PARSE input WITH g AT root</line3>
            <line3>IF error.free:
               <line4>SERIALISE root FROM trace WITH g USING input</line4>
            </line3>
         </line2>
         <line2>ELSE:
            <line3>WRITE "Failed" /</line3>
         </line2>
      </line1>
   </how-to>
```

34 **Use of XForms**

The code is read in (in two stages, one for each part), and then displayed.

```
<repeat ref="*">
   <group ref=".[name()='how-to']">
      <switch>
         <case id="closed">
               displays just the header
         <case id="open">
               displays the header and content
         </case>
      </switch>
   </group>
   ...similar treatment for documentation...
</repeat>
```

35 **Header**

```
<trigger appearance="minimal">
   <label>
      <output class="header-closed" ref="header"/>
      <output class="comment1" ref="comment"/>
   </label>
   <toggle case="open" ev:event="DOMActivate"/>
</trigger>
```

36 **Header+Content**

```
<trigger appearance="minimal">
   <label>
      <output class="header-open" ref="header"/>
      <output class="comment1" ref="comment"/>
   </label>
   <toggle case="closed" ev:event="DOMActivate"/>
</trigger>
<repeat ref="descendant::*">
   <output class="{name()}" ref="text()"/>
   <output class="comment1" ref="comment"/>
</repeat>
```

37 **Search interface**

A search string:

```
<input incremental="true" ref="q"><label>search</label></input>
```

And only repeat over matching elements:

```
<repeat ref="*[contains(., instance('q')/q)]">
```

38 **Demo**

ixampl

39 **Adaptation for other languages**

There's nothing essential to ABC in the browser. All there is is folding sections of two types.

Each fold consists of a header and a number of lines at different indentations, with optional comments.

So changing the abstractions suitably, any format can be injected into the browser, to make a quick and easy code browser.

40 **Code size**

The browser uses 3 declarative languages:

- ixml: 29 lines
- xforms: 87 lines
- css: 25 lines

41 **Conclusion**

ixml makes injection of unstructured data into the XML pipeline easy!

Declarative languages make life easy

42 **Thank you**

43 **References**

Unicode in 16 bits:

Becker, Joseph D. (1998-09-10) [1988-08-29]. "Unicode 88" (PDF). unicode.org. Unicode Consortium. https://unicode.org/history/unicode88.pdf

UTF-8:

https://www.cl.cam.ac.uk/~mgk25/ucs/utf-8-history.txt

https://www.cl.cam.ac.uk/~mgk25/ucs/UTF-8-Plan9-paper.pdf

ixml:

https://invisiblexml.org/

http://invisiblexml.org/1.0/

ixampl:

https://archive.xmlprague.cz/2022/files/xmlprague-2022-proceedings.pdf#page=51