xml2tex

The easy way to define XML-to-LaTeX converters

Created by Keiichiro Shikano / @golden_lucky

What xml2tex is NOT.

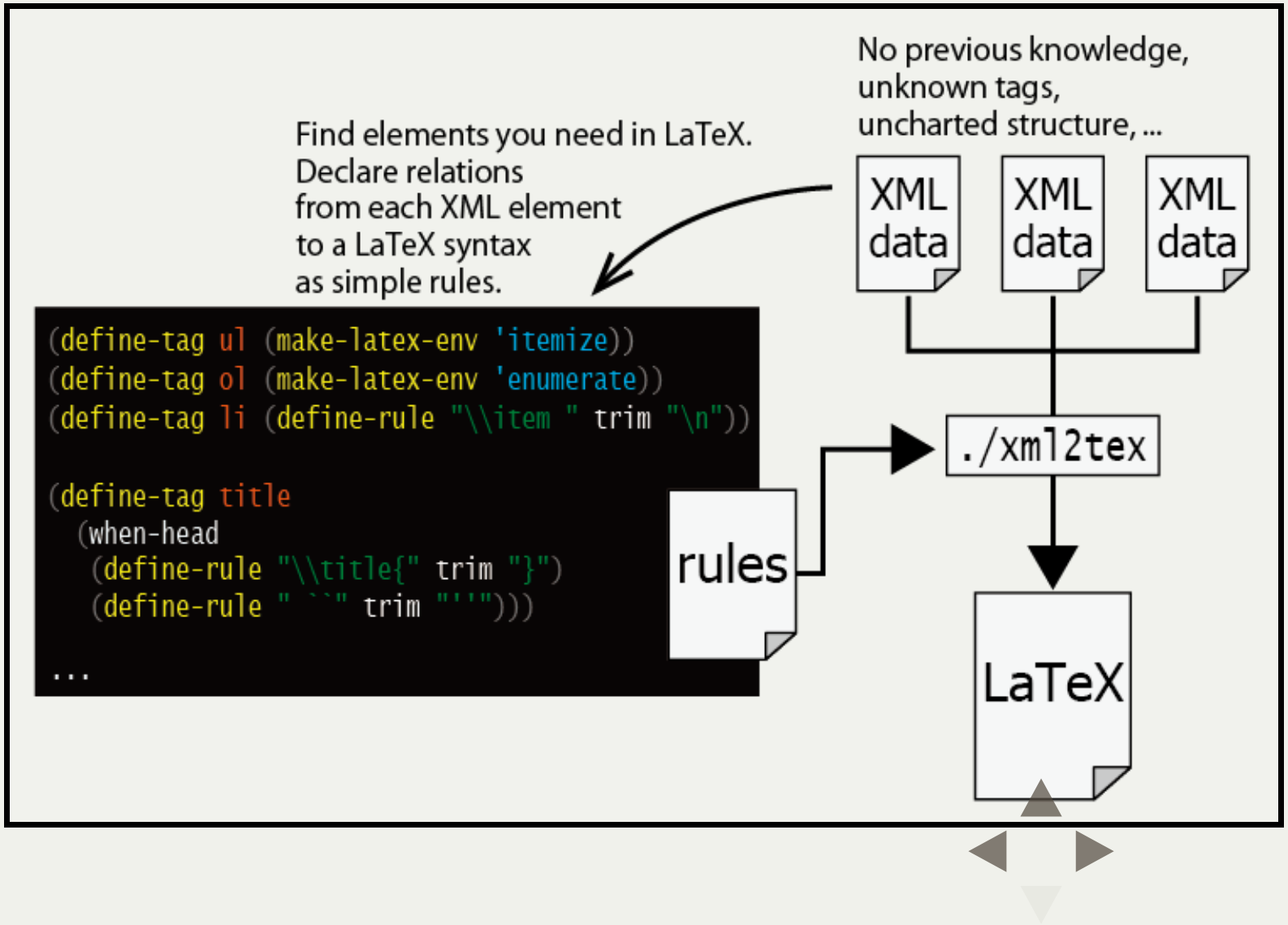
- xml2tex is NOT a markup.
- xml2tex is NOT a ready-to-use converter application.

xml2tex is a framework to give XML syntax a nice presentation layer using LaTeX.

xml2tex is a framework to give XML syntax a nice presentation layer using LaTeX.

xml2tex is a framework for using XML syntax as a source of LaTeX.

Overview of xml2tex

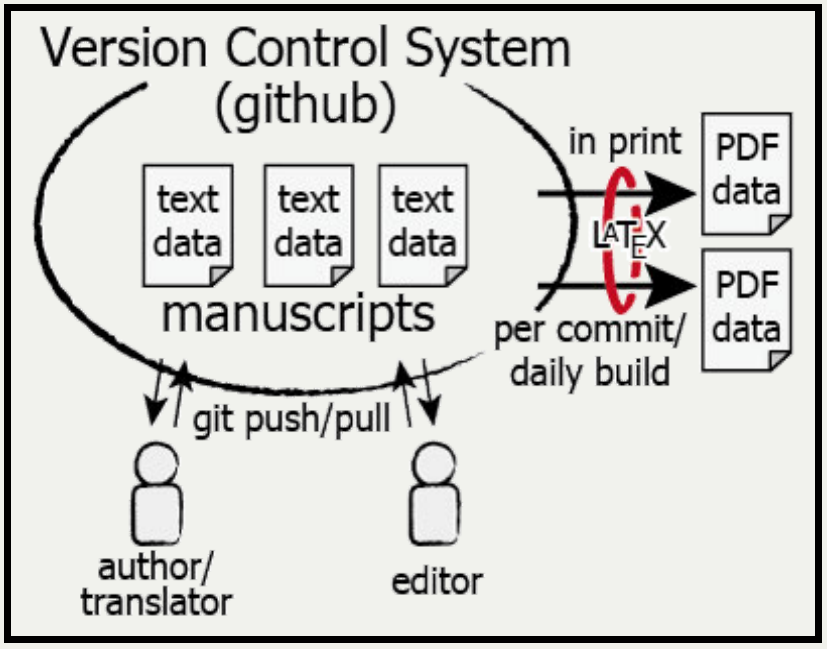


Accusation

”Are you idiot?
Why on earth are you going to use ugly XML syntax instead some more concise syntax?“

Our Way of Creating a Book

- First of all, we don't want to use WYSIWYG application to create a book being sold.



- We usually maintain manuscripts using VCS (github) to the very last.
- It makes working together smoother over the Internet.
- LaTeX is one of the best tools for typesetting in this kind of environment.

- Taking advantage of *xml2tex*, we could use XML as manuscripts.

Alternative approaches

- Use *XSLT* and *XSL-FO* for typesetting.
- Use *XSLT* for getting *LaTeX*.
- Convert them into *LaTeX (once and for all)*.
- Or better yet, convert them into other *standard markups* or *markdowns* (and use the corresponding environment like DocBook, Sphinx, pandoc, TeXML, and so on).

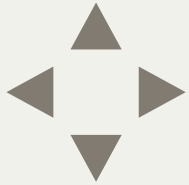
Pros of using XML (which LaTeX also has)

- We need a variety of *meta data* to create a book; editor's comments, index entries, and original texts.
- We need to be able to achieve *rich enough page layout* for a commercial book.

A real example of XML source for translating projects

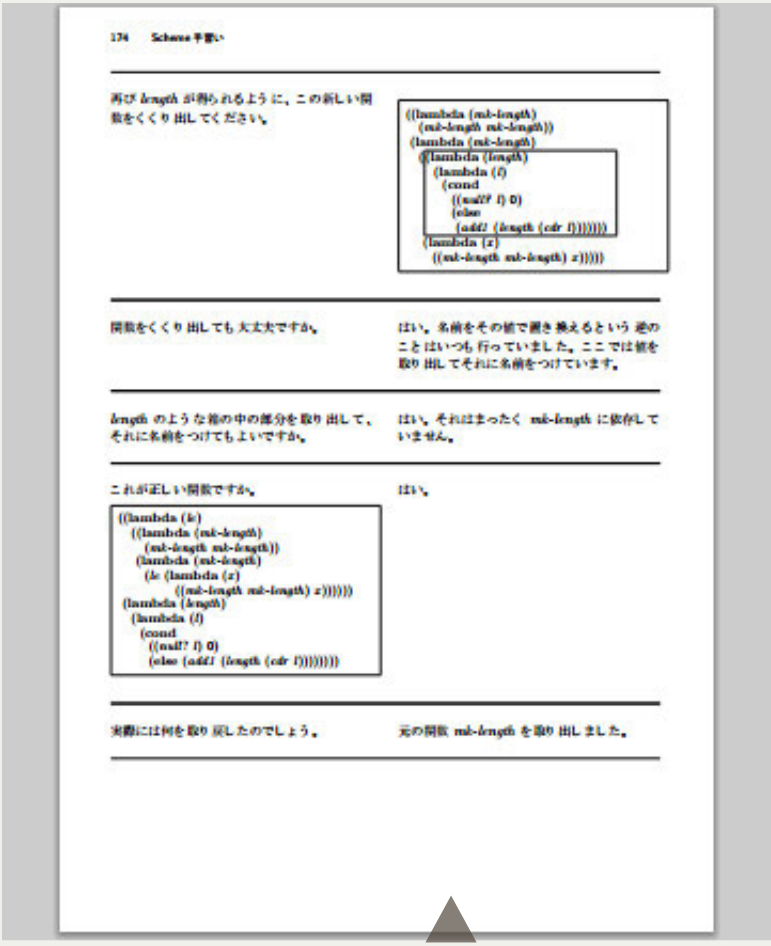
We usually let the translator put down the corresponding Japanese text below each original paragraphs like this.

```
<title lang="en">Lorem ipsum</title>
<title lang="en">いろはにほへと</title>

<p lang="en">dolor sit amet, </p>
<p lang="ja">ちりぬるを</p>

<blockquote lang="en">consectetur adipisicing elit.</blockquote>
<blockquote lang="ja">わかよたれそつねならむ。</blockquote>
```



A real example of unusual page layout

```
<left lang="en">
Move out the new function so that we get /length back.
</left>
<left lang="ja">
再び /length が得られるように、
この新しい関数をくくり出してください。
</left>
<right lang="en">
<program>
((lambda (/mk-length)
   (/mk-length /mk-length))
 (lambda (/mk-length)
   ([(lambda (/length)                    ]
    [  (lambda (/l)                       ]
    ...
    [        (/add1 (/length (/cdr /l)))))))]
    (lambda (/x)
```



174 Scheme 手習い

再び length が得られるように、この新しい関数をくくり出してください。

```
((lambda (mk-length)
   (mk-length mk-length))
 (lambda (mk-length)
   ((lambda (length)
      (lambda (l)
        (cond
          ((null? l) 0)
          (else
            (add1 (length (cdr l)))))))
    (lambda (x)
      ((mk-length mk-length) x)))))
```

関数をくくり出しても大丈夫ですか。

はい。名前をその値で置き換えるという逆のことはいつも行っていました。ここでは値を取り出してそれに名前をつけています。

length のような値の中の部分を取り出して、それに名前をつけてもよいですか。

はい。それはまったく mk-length に依存していません。

これが正しい関数ですか。

はい。

```
((lambda (le)
   ((lambda (mk-length)
      (mk-length mk-length))
    (lambda (mk-length)
      (le (lambda (x)
            ((mk-length mk-length) x))))))
 (lambda (length)
   (lambda (l)
     (cond
       ((null? l) 0)
       (else (add1 (length (cdr l))))))))
```

実際には何を取り戻したのでしょう。

元の関数 mk-length を取り出しました。

A real example of using meta data

再び *length* が得られるように、この新しい関数をくくり出してください。

```
((lambda (mk-length)
   (mk-length mk-length))
 (lambda (mk-length)
   ((lambda (length)
      (lambda (l)
        (cond
          ((null? l) 0)
          (else
           (add1 (length (cdr l)))))))
    (lambda (x)
      ((mk-length mk-length) x)))))
```

Move out the new function so that we get *length* back.

```
((lambda (mk-length)
   (mk-length mk-length))
 (lambda (mk-length)
   ((lambda (length)
      (lambda (l)
        (cond
          ((null? l) 0)
          (else
           (add1 (length (cdr l)))))))
    (lambda (x)
      ((mk-length mk-length) x)))))
```

関数をくくり出しても大丈夫ですか。

はい。名前をその値で置き換えるという逆のことはいつも行っていました。ここでは値を取り出してそれに名前をつけています。

Is it okay to move out the function?

Yes, we just always did the opposite by replacing a name with its value. Here we extract a value and give it a name.

length のような箱の中の部分を取り出して、それに名前をつけてもよいですか。

はい。それはまったく *mk-length* に依存していません。

Can we extract the function in the box that looks like *length* and give it a name?

Yes, it does not depend on *mk-length* at all!

これが正しい関数ですか。

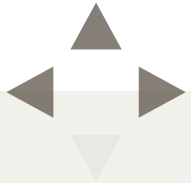
```
((lambda (le)
   ((lambda (mk-length)
      (mk-length mk-length))
    (lambda (mk-length)
      (le (lambda (x)
            ((mk-length mk-length) x))))))
 (lambda (length)
   (lambda (l)
     (cond
       ((null? l) 0)
       (else (add1 (length (cdr l))))))))
```

はい。

Is this the right function?

```
((lambda (le)
   ((lambda (mk-length)
      (mk-length mk-length))
    (lambda (mk-length)
      (le (lambda (x)
            ((mk-length mk-length) x))))))
 (lambda (length)
   (lambda (l)
     (cond
       ((null? l) 0)
       (else (add1 (length (cdr l))))))))
```

Yes.



Pros of using XML, instead of LaTeX

- Authors of technical books tend to have HTML literacy, provided that there's *no excessive information* against human.
- Easy to confirm the appearance of structured text just by Web browsers, provided that there's an *appropriate CSS*.
- It's *technically possible* to generate EPUB as well as PDF.

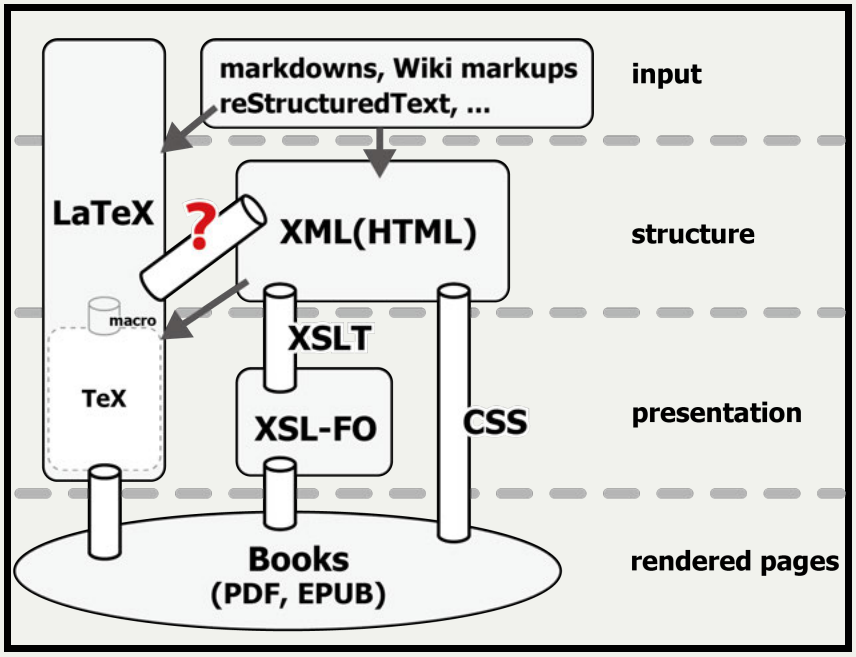
Cons of using XML

- If there's *excessive information* just for machine, it becomes hard to edit the manuscript.
- If there's no *appropriate CSS*, you have to rely only on the abstract structure of documents.
- It's *technically possible*, but not trivial to generate EPUB.

More to the point...

- Creating PDFs from XML often requires some *proprietary software* and *XSLT*!

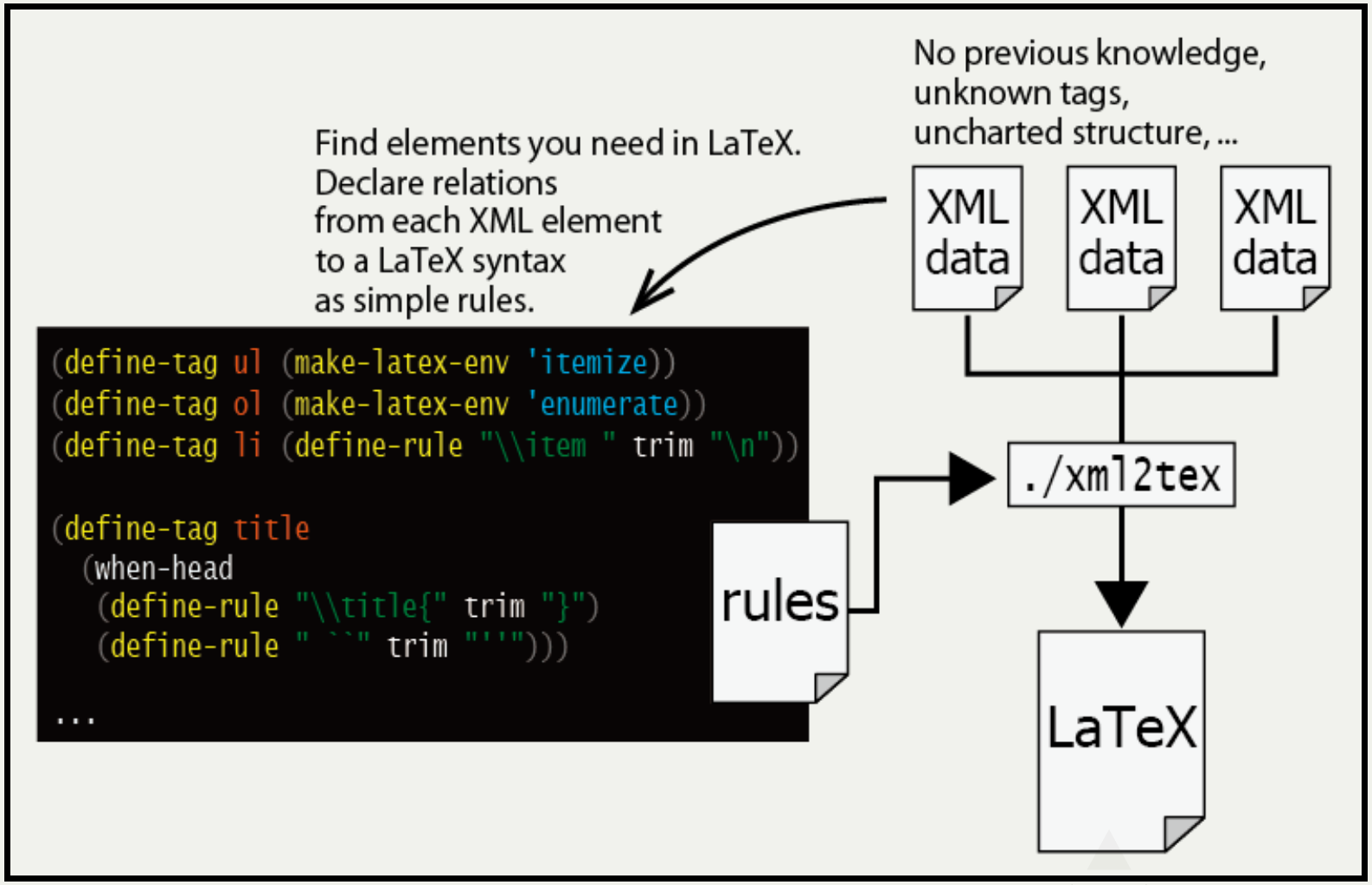
What we need is an *easy* way to get XML-to-LaTeX converter, *as needed*.



- Without any prior information for the structure,
- Without any restriction for the structure,
- Without explicitly writing XML parser every time,

- With a profound support for generating LaTeX documents. (XSLT won't be the best solution!)

xml2tex — our approach



How to get a LaTeX representation of this XML?

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="book.css" type="text/css" charset="UTF-8"?>
<XML>
<TITLE>A Great Book</TITLE>
<Chapter-Title>Writing in Practice</Chapter-Title>
<Body-Text-First>Before you can start writing a real book ... </Body-Text-First>
<Body-Text>Let&rsquo;s get started!</Body-Text>

<Heading-1>Introduction to LaTeX</Heading-1>
<Body-Text-First>To keep on attracting your readers...</Body-Text-First>

<Code-First>$ latex </Code-First>
<Code>This is pdfTeX, Version 3.14159265-2.6-1.40.15 (TeX Live 2014) (preloaded format=latex)</Code
<Code> restricted ¥write18 enabled.</Code>
<Code-Last>**</Code-Last>

<Body-Text>Note that you can&rsquo;t put down % in your masterpiece.  </Body-Text>
```

Note that it was *converted* from a DTP application.

Things you can tell just by looking at the XML

- It at least has the elements named <TITLE>, <Chapter-Title>, <Body-Text-First>, <Body-Text>, <Heading-1>, <Code-First>, <Code>, and <Code-Last>.
- It at least has a XML character entity ’.
- <?xml...?> is the processing instruction, which generally seems to be useless in LaTeX.
- Characters like %, $ and ¥ may need to be escaped.

All you need to tell xml2tex

```
(define-tag XML    (make-latex-env 'document))

(define-tag TITLE (make-latex-cmd 'title))

(define-tag Chapter-Title  (make-latex-cmd 'chapter))
(define-tag Heading-1      (make-latex-cmd 'section))

(define-tag Body-Text-First (define-rule "¥n¥¥noindent{}" trim "¥¥par¥n"))
(define-tag Body-Text       (define-rule "¥n" trim "¥¥par¥n"))

(define-tag Code-First (define-rule "¥¥begin{alltt}" kick-comment ""))
(define-tag Code       (define-rule "" kick-comment ""))
(define-tag Code-Last  (define-rule "" kick-comment "¥¥end{alltt}"))
```

That's it!

Save these lines into a file, and it can be used as a kind of specification for xml2tex.

Converting XML with the rule file

```
$ xml2tex -r demo.rules demo.xml > demo.tex
```

where demo.rules is the rule file defined before.

```
¥documentclass{book}
¥usepackage[T1]{fontenc}
¥usepackage{alltt}

¥begin{document}
¥title{A Great Book}

¥chapter{Writing in Practice}

¥noindent{}Before you can start writing a real book ... ¥par


Let's get started!¥par
```

pdfLaTeX result

Chapter 1

Writing in Practice

Before you can start writing a real book ...
Let's get started!

1.1 Introduction to LaTeX

To keep on attracting your readers...

```
$ latex
This is pdfTeX, Version 3.14159265-2.6-1.40.15 (TeX Live 2014) (preloaded format=latex)
 restricted \write18 enabled.
**
```

Note that you can't put down % in your masterpiece ...

Generating LaTeX syntax from the document tree is defined as a rule.

```
(define-rule
  "¥n"          ; Put this at the beginning.
  trim          ; Its text nodes should be treated with this.
  "¥¥par¥n"))   ; Put this at the ending.
```

1. Preceding string, or a thunk which returns a preceding string. Possible example is `¥¥texttt{`.
2. A procedure from string to another string. `trim` is one of such procedures. It takes a string and returns a string in which special characters in LaTeX are escaped properly.
3. Following string, or a thunk which returns a following string. Possible example is `}`.

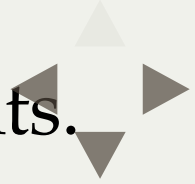
Let the defined rule map the content from XML element to LaTeX syntax, based on the tag name.

```
(define-tag Body-Text     ; If the XML node has this name, ...
  (define-rule            ; Apply this rule to the content.
    "¥n"
    trim
    "¥¥par¥n"))
```

Just putting down these definitions for each XML tags is enough to convert the XML into LaTeX.

Supportive features in defining rules

- `(make-latex-cmd 'cmdname)`
  generates a rule for creating a LaTeX command ¥¥cmdname{...} with the contents.
- `(make-latex-env 'envname)`
  generates a rule for creating a LaTeX environment ¥¥begin{envname}...¥¥ent{envname} with the contents.
- `(through)`
  generates a rule for putting down all the contents with necessary escaping.
- `(ignore)`
  generates a rule for discarding the contents.



Default rule is "through"

```
$ xml2tex -r my.rule input.xml
Not knowing the LaTeX syntax for <div>, ... applyed (through).
Not knowing the LaTeX syntax for <div>, ... applyed (through).
Not knowing the LaTeX syntax for <div>, ... applyed (through).
Not knowing the LaTeX syntax for <div>, ... applyed (through).
Not knowing the LaTeX syntax for <div>, ... applyed (through).

¥documentclass{book}
¥usepackage[T1]{fontenc}
¥begin{document}
 ¥chapter{Starting Out}

..¥par
...
```

The elements you haven't define any explicit rule are indicated while the conversion. It helps you try detecting the unknown elements within the given XML file.

Supportive features in adoring tree

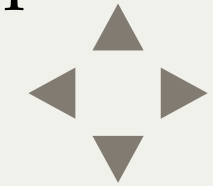
- `($parent? '(tag1 tag2 ...))`
  returns True if the node is directly under `tag1` or `tag2` ...

  There's a lot more similar functions like `$parent` (takes the parent name), `$siblings?`, `$under?`, and so on.
- `($@ 'attrname)`
  returns a string value if the node has an attribute of the `attrname`.

An example of $parent and $parent?

```
(define-tag title
  (define-rule
    (lambda ()
     (cond
      (($parent? '(chapter))      "¥¥chapter{")
      (($parent? '(sect1))        "¥¥section{")
      (($parent? '(sect2 sect3)) "¥¥subsection{")
      (else (error "no rule for title" ($parent)))))
    trim
    "}"))
```

- $-functions can be used within (define-rule ....
- Also note that (define-rule ... takes a procedure instead of a fixed string for its first argument.



An example of $@ (getting attribute value)

```
(define-tag img
  (define-rule
    (list "¥¥begin{figure}¥n"
          "¥¥includegraphics"
          #`"[width=,($@ 'width)]"
          #`"{,($@ 'src)}")))
    trim
    "¥¥end{figure}"))
```

- Note that #`"..." is a syntax for a string interpolation. It actually a feature of Gauche, a Scheme programming language on which xml2tex works.

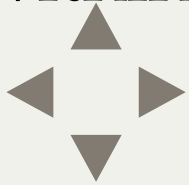
More practical example — HTML tables

```
<body>
<table>
  <tr>
    <td bgcolor="#ff0000" width="30%">1</td>
    <td bgcolor="#33cccc" width="20%">2</td>
    <td bgcolor="#00ff00">3</td>
    <td rowspan="2" bgcolor="#ff9900">4</td>
  </tr>
  <tr>
    <td bgcolor="#00ffff">A</td>
    <td bgcolor="#ff00ff" align="right" colspan="2" rowspan="2" width="50%">B</td>
  </tr>
  <tr>
    <td bgcolor="#ffff00">C</td><td bgcolor="#00cc33" width="20%">D</td>
  </tr>
</table>
</body>
```

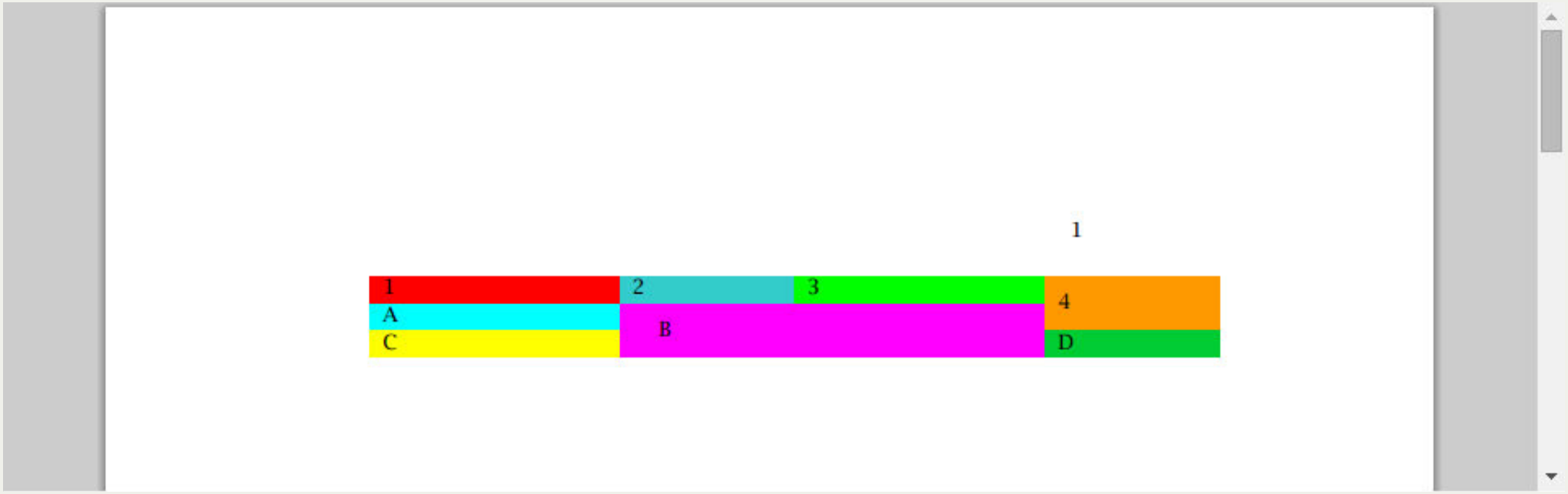
Possible rule to convert HTML table to LaTeX's tabular environment

```
(define-tag table
  (define-rule
    #`"¥¥begin{tabular}{|,($@ 'colspec)|}¥n" ; colspec is a generated attribute
    trim
    "¥¥end{tabular}"

    :pre
    (lambda (body root)
      (let* ((trs ((node-closure (ntype-names?? '(tr))) body))
             (tds (map (node-closure (ntype-names?? '(td th))) trs))
             (tr-attrs (map sxml:attr-as-list trs))
             (colspec (make-colspec tds)))
        (sxml:set-attr
```

- It requires transformation of the tree, before applying a rule defined by `define-rule`.
- To transform the tree, use `:pre` keyword within `define-rule`.



Results



1
2
3
4
A
B
C
D

Practical example 2 — Footnotes

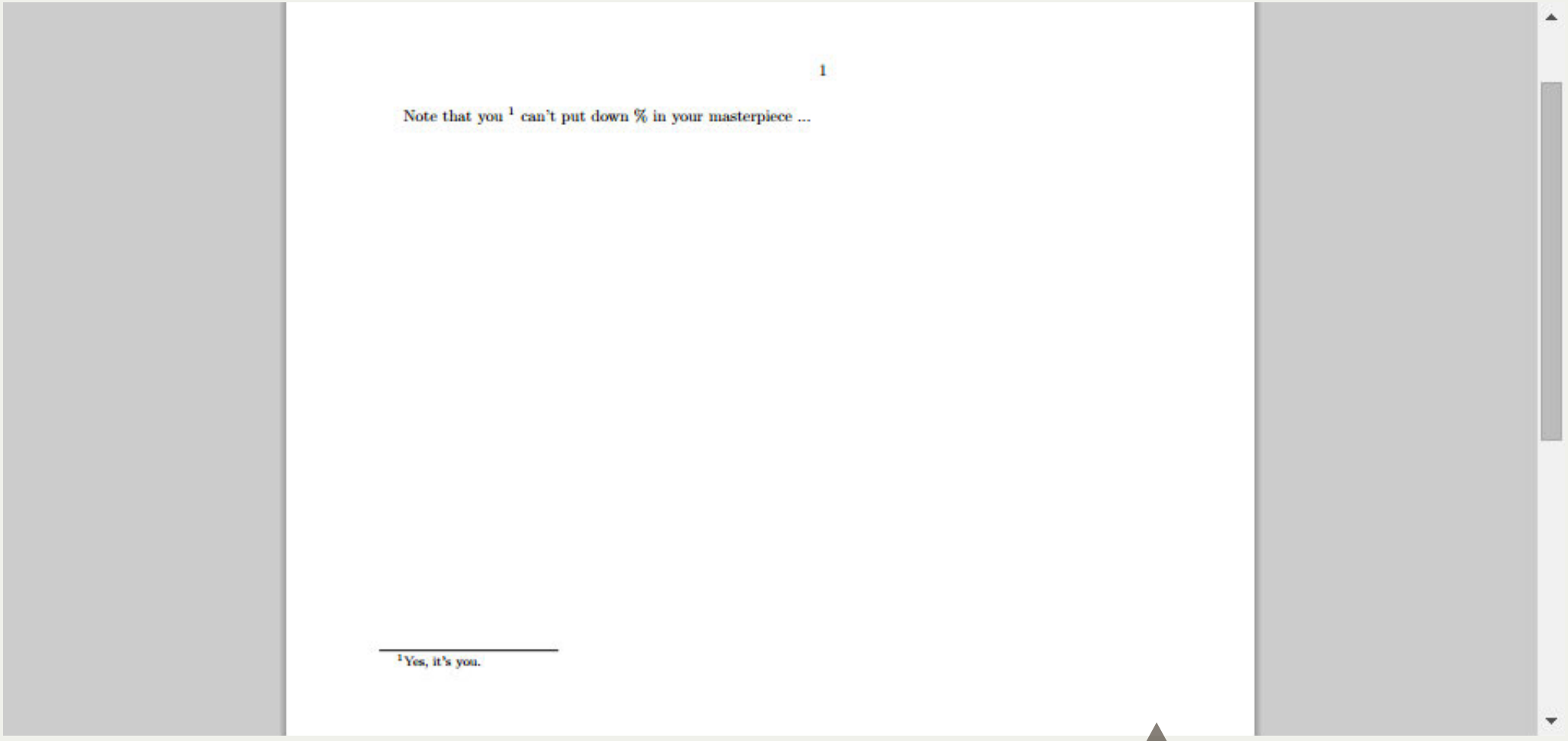
```
<XML>
<Body-Text>Note that you <A href="#pgfId-1018223" CLASS="footnote">1</A>
  can't put down % in your masterpiece ...</Body-Text>

<FOOTNOTES>
<FOOTNOTE>
<Footnote-Text><A ID="pgfId-1018223"></A>
Yes, it's you.
</Footnote-Text>
</FOOTNOTE>
</FOOTNOTES>
</XML>
```

Possible rule to make LaTeX's ¥footnote from <FOOTNOTES> at the bottom

```
(define-tag A
  (define-rule "" trim ""
    :pre
    (lambda (b r)
      (cons 'A
            (map-union (lambda (e) (map-union
              (lambda (a)
                (if (string=?
                      (ifstr (sxml:attr-u b 'href))
                      #`"#,(sxml:attr-u a 'ID)")
                    e #f))
              ((select-kids (ntype-names?? '(A))) e)))
            ((node-closure (ntype-names?? '(Footnote-Text))) root))))))
```

- This time you need to get hold of a subtree, and attach it to another node.
- Things has been rather messy, but :pre still works.

Results



1

Note that you [1] can't put down % in your masterpiece ...

[1] Yes, it's you.

Conclusion

- XML is not bad for making books.
  - Simple markups/markdowns won't offer you a rich layout.
  - LaTeX will provide a concrete presentation layer for XML documents.
    - However, using XSLT for converting XML to LaTeX is a hard way, because XSLT is a tool for getting another XML from a XML.
- One major missing-link is a complete and lightweight way of getting LaTeX from XML.
  - *xml2tex* would be a solution with its declarative style of defining whatever conversion rules.
  - ConTeXt's XML support is probably another solution.
- It requires some experiences on XML, as well as LaTeX.
  - In addition to that, xml2tex requires Scheme experience.