Why our HTML Docs don’t just `Print` and what to do about it

Nikolaj Potashnikov

PhD in Economics, Solution architect, Course-IT

consulting@yandex.ru @nmpotashnikoff

1. Way in converting simple text markup to `print` formats

- https://github.com/CourseOrchestra/course-doc: XSL-FO templates for Asciidoctor → Docbook backend
- https://github.com/CourseOrchestra/asciidoctor-open-document: Open Document Converter for Asciidoc
- https://github.com/fiddlededee/unidoc-publisher: UniDoc Publisher — any markup to any printing rendering engine

2. `Print`, `Export to Word`, `Export to PDF` are very often just a trap

What to do with long line in listing?

- We may scale
- Or use landscape orientation
- Or both, but would it be enough?
- If not, we may fire error for long lines
- Or wrap them
- With linefeed and spaces? And how to copy?
- With indents? Still impossible to copy from PDF

And in web we can just add horizontal scroll bar.

3. Main formats for printing

1. PDF
2. Text processing formats (Open XML — MS Office, Open Document — LibreOffice)
3. HTML?

CSS Paged Media — CSS extension, defining style specific for printing

3.1. Most widespread rendering approaches

- PDF ← native PDF-generating libraries
- PDF ← XSL-FO with FOP-processors
- PDF ← via TeX
- PDF ← HTML + Paged Media CSS
- DOCX/ODT, PDF ← +/- text processors (MS Word, LO Writer)

These technologies are not aligned in infinite details like:

- Apache FOP has problems with Leader alignment (dots in a table of contents)
- LO Writer doesn't support typography (like keep with next) within table cells
- Microsoft doesn't recommend to run automation tasks (like saving PDF) on a server

4. Some brief conclusions

Feel like speleologist?

- The world of printing is the world of constraints
- And that constraints differ for each technology, you often need to support several chains (exquisitely looking PDF with TeX and LibreOffice for coordination)
- With no universal solutions

5. UniDoc Publisher approach suits best if at least one of

- You don’t prepare documentation especially for printing purposes
- You are automating documentation generation and hope it will look good, no matter what will be generated
- You output format is one of text processing format

6. In search for flexibility: Asciidoctor open document

Automation on the writer side

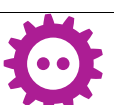
1. Asciidoctor parses markup into AST (Abstract Syntax Tree)
2. You may transform AST with Asciidoctor AST processing
3. Asciidoctor runs writer template for each AST node recursively
4. You may write you code in pure Ruby or with special Slim templates

6.1. A simplified processing AST example

```
- !<OrderedList>
  roles:
  - "arabic"
  id: "ol-1"
  captioned_title:
      children:
      - !<Text>
        text: "Automation"
  children:
  - !<ListItem>
    children:
    - !<Paragraph>
      children:
      - !<Text>
        text: "Asciidoctor..."
  - !<ListItem>
...
```

```
- list_style = "#{get_basic_style}"
- if captioned_title?
  text:p text:style-name="#{list_style}"
    text:bookmark text:name="#{id}"
    =captioned_title

text:list text:style-name="#{list_style}"
  - items.each_with_index do |item, index|
```

6.2. Great, but

- You can’t override part of a template
- You should invent styling approach

Styling? But text processors do support styling!

- `<span class="bold green">bold, green</span>` – impossible to apply two styles to one element

- Asciidoctor Open Document introduces some extended Open Document format to preserve Asciidoctor AST contents
- Each function checks, if style should be applied, and if yes, applies it

6.3. And still

- Unexpectedly transforming this extended Open Document format became one of the most used feature of Asciidoctor Open Document
- Styling as separate task of writing proved also to be useful
- The Gradle was magnificent in gluing all parts together

7. Thoughts before the second step

- If creating universal converter impossible…
- We should create meta converter — platform for building converters

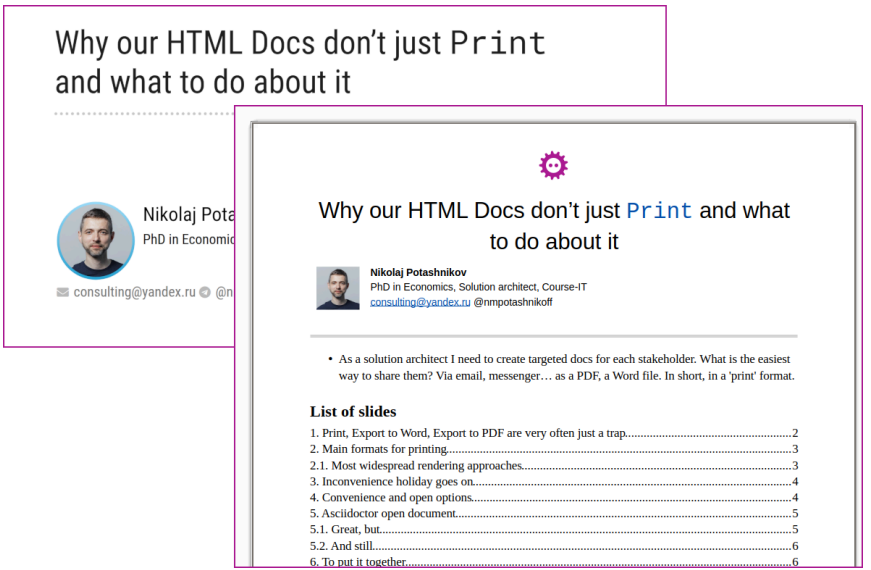
Estimated requirements

- Native converter as a reader
- Great ways of transforming AST
- A good approach for styling as a separate focus
- 99% generic writer
- Good integration with CI/CD

8. Native converter as a reader?

Each converter outputs HTML. HTML is quite semantic, why shouldn't we use it?

9. Let’s convert this presentation to LO Writer



Why our HTML Docs don’t just Print and what to do about it

10. Notes on this demo

- Everything is in a single `build.gradle.kts` file
- All Kotlin code examples are just includes from this `build.gradle.kts`

11. Boilerplate

```
FodtConverter {
    html = AsciidocHtmlFactory()
        .getHtmlFromFile(File("${project.projectDir}/$presentationFile.adoc"), true)
    template = File("${project.projectDir}/template-1.fodt").readText()
    adaptWith(AsciidoctorOdAdapter)
    unknownTagProcessingRule = unknownTagProcessingRuleRevealJs()
    parse()
        // Processing AST
    ast2fodt()
    File("${project.projectDir}/output/ast.yaml").writeText(ast().toYamlString())
    File("${project.projectDir}/output/$presentationFile-notes-v$version.fodt").writeTe
}
```


12. Processing AST

```
ast().descendant { section ->
    section.sourceTagName == "section" &&
            section.descendant { it is Heading && it.level == 1 }
                .isNotEmpty()
}.first().also { it.insertBefore(makeTitle(it)) }.remove() (1)
```


13. Rearranging title (Asciidoc source)

```
|===
a|
[.title-photo]
image::images/nmp1.jpg[]
a|
[.full-name]
Nikolaj Potashnikov

[.bio]
PhD in Economics, Solution architect, Course-IT
.2+>.>a|{nbsp}
[.logo]
image::images/fosdem-logo.svg[]
2+a|
[.contact]
icon:envelope[] consulting@yandex.ru icon:telegram[]{nbsp}@nmpotashnikoff
|===
```


14. Rearranging title, extracting semantics

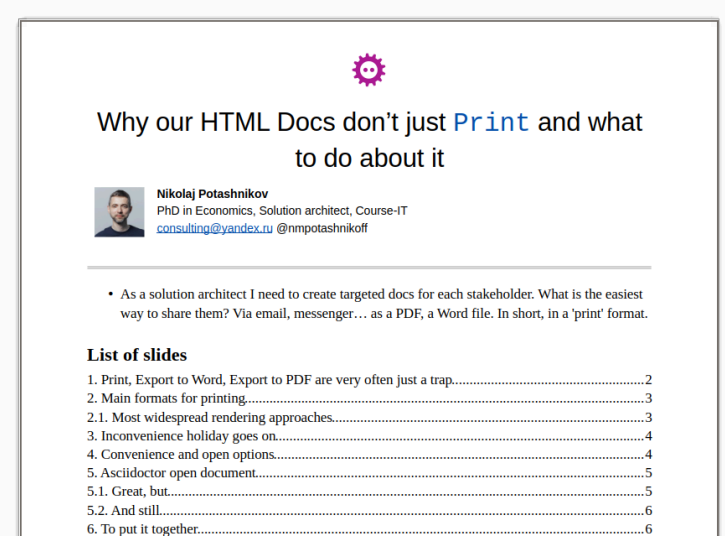
```
val title = sourceNode.descendant { it is Heading && it.level == 1 }.first()
val notes = sourceNode.descendant { it.sourceTagName == "aside" }.first()
val (fullName, bio, photo, contact, logo) =
    arrayOf("full-name", "bio", "title-photo", "contact", "logo")
        .map { role -> sourceNode.descendant { it.roles.contains(role) }.first() }

logo.descendant { it is Image }.first().let { it as Image }
    .width = Length(1000F, LengthUnit.cmm)
photo.descendant { it is Image }.first().let { it as Image }
    .width = Length(1500F, LengthUnit.cmm)
```


15. Rearranging title, constructing title

```
appendChild(logo)
appendChild(title)
table {
    col(Length(18F)); col(Length(152F))
    roles("about-me")
    tableRowGroup(TRG.body) {
        tr {
            td { appendChild(photo) }
            td { arrayOf(fullName, bio, contact).forEach { appendChild(it) } }
        }
    }
}
appendChild(notes)
appendChild(Toc(2, "List of slides"))
normalizeImageDimensions()
```


16. Let’s return to the result



Why our HTML Docs don’t just Print and what to do about it

Why our HTML Docs don’t just Print and what to do about it

Nikolaj Potashnikov
PhD in Economics, Solution architect, Course-IT
consulting@yandex.ru @nmpotashnikoff

- As a solution architect I need to create targeted docs for each stakeholder. What is the easiest way to share them? Via email, messenger… as a PDF, a Word file. In short, in a 'print' format.

List of slides

17. And a little bit of styling

```
OdtStyle { p ->
    if (p !is Paragraph) return@OdtStyle
    if (p.ancestor { it.roles.contains("logo") }.isEmpty()) return@OdtStyle
    attributes("style:master-page-name" to "First_20_Page")
},
OdtStyle { tableCell ->
    if (tableCell !is TableCell) return@OdtStyle
    if (tableCell.ancestor { it.roles.contains("about-me") }.isEmpty()) return@OdtStyle
    tableCellProperties {
        arrayOf("top", "right", "bottom", "left")
            .forEach { attributes("fo:border-$it" to "none") }
    }
},
```


18. Let's return to processing AST

```
ast().descendant { it.roles.contains("notes") } 1
    .forEach { it.insertBefore(HorizontalLine()) }
ast().descendant { it is Heading && it.level > 1 }
    .forEach {
        it.insertBefore(
            Paragraph().apply { roles("slide-finish") }
        )
    } 2
odtStyleList.add(odtStyles())
odtStyleList.add(rougeStyles()) 3
```


19. Extending AST

```
class HorizontalLine() : NoWriterNode() {
    override val isInline: Boolean get() = false
}
```

```
OdtCustomWriter { horizontalLine ->
    if (horizontalLine !is HorizontalLine) return@OdtCustomWriter
    preOdNode.apply {
        "text:p" {
            attributes("text:style-name" to "Horizontal Line")
            process(horizontalLine)
        }
    }
},
```


19.1. Testing

Content type	The result
paragraph	Paragraph 1 Paragraph 2
list	1. Item 1 1. Subitem 1. Subitem 1. Subitem 2. Item 2
table	Subtable

Some paragraph after table.

Some paragraph after table.

Why our HTML Docs don't just Print and what to do about it

20. Conclusion

- If you need to get doc in printing format, don’t pursue universality, limit, limit and limit
- Don’t search for solution with options, prepare to code
- Ecosystem is crucial. Gradle + Kotlin stack is just a coincidence, where syntax, wide ecosystem and CI accidentally met
- Still more coincidence, they met Asciidoctor
- Still more coincidence, they met LibreOffice API
- Too much coincidence for coincidence?

21. Questions?

Why our HTML Docs don't just Print and what to do about it

Why our HTML Docs don't just Print and what to do about it

22.1. Comparison

Technology	Strengths	Weaknesses
Native	• The technology is typically used by converters with a low entry barrier	• Most often has a unique configuration approach • High likelihood of encountering unexpected limitations
XSL-FO	• Time-tested technology, W3C standard	• Different processors yield different results (e.g., varying typography limitations) • Better to stick to FO; XSLT is outdated and hard to maintain • Slow build performance (at least with Apache FOP)

23. Comparison #2

Technology	Strengths	Weaknesses
TeX	• Best-in-class typography, suitable for mass printing • Mature, albeit complex, ecosystem	• …albeit complex ecosystem
Paged Media CSS	• W3C standard • Takes HTML as input, so it's practically applicable everywhere	• Browser support remains very limited • Among free tools, I would recommend only WeasyPrint

24. Comparison #3

Technology	Strengths	Weaknesses
Word processors	• Format is designed for manual editing of the output document • Fastest way to produce output (even when PDF is required)	• Only LibreOffice aligns with the DocOps paradigm • Imposes numerous constraints, including on typography • Saving to a word processor format almost always requires extensive custom adjustments and limitations