

Presentation of XML

Patryk Czarnik

XML and Applications 2016/2017

Lecture 11 - 26.05.2017

Separation of content and formatting

- According to best XML practices:
 - Documents consist of content / data.
 - Tags are for structure and meaning (semantic tagging).
 - e.g. `<amount>2.99</amount>` rather than `<i>2.99</i>`
 - There is no direct formatting information.
- How to present documents?
 - Generic (and poor) XML presentation methods
 - XML source
 - document tree
 - unformatted text content
 - Custom application handling a particular known class of documents
 - Importing XML to text editors or DTP tools
 - External style sheets

Idea of stylesheet

```
<person position="expert" id="102103">  
  <fname>Dawid</fname><surname>Paszkiwicz</surname>  
  <phone type="office">+48223213203</phone>  
  <phone type="mobile">+48501502503</phone>  
  <email>paszkiewicz@example.com</email>  
</person>
```

```
<person position="expert" id="102105">  
  <fname>Marek</fname><surname>Kacki</surname>  
  <phone type="office">+48223213212</phone>  
  <phone type="mobile">+48501502524</phone>  
  <email>kacki@example.com</email>  
</person>
```

- white background, blue frame
- font 'Times 10pt'
- 12pt for name
- abbreviation before phone number
- email in italic
- position before name

- yellow background, blue 3D frame
- font 'Bookman 12pt'
- phone numbers in single line
- email in typewriter font
- no too much details

expert
Dawid Paszkiewicz
tel. +48223213203
mob. +48501502503
paszkiewicz@example.com

assistant
Marek Kacki
tel. +48223213212
mob. +48501502524
kacki@example.com

Dawid Paszkiewicz
+48223213203 +48501502503
paszkiewicz@example.com

Marek Kacki
+48223213212 +48501502524
kacki@example.com

Benefits of content and formatting separation

- With semantic tagging – source data analysis easier and more reliable (than reverse-engineering of formatted text)
- Ability to easily present
 - the same document after modifications
 - other documents from the same class
- Changes in formatting applied easily
 - modifications in one place – the stylesheet
 - whole class of documents formatted consistently
- Alternative styles for the same class of documents, depending on
 - media type (screen, printout, voice)
 - details level
 - reader preferences (or disabilities...)

Standards related to XML presentation

- Assigning style to document:
 - *Associating Style Sheets with XML documents*
- Stylesheet languages:
 - DSSSL (historical, used for SGML)
Document Style Semantics and Specification Language
 - CSS
Cascading Style Sheets
 - XSL
Extensible Stylesheet Language

Associating style with document

- Using `xml-stylesheet` processing instruction
- Defined in W3C recommendation
Associating Style Sheets with XML documents

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet
  type="text/css" href="blue.css" ?>
<?xml-stylesheet title="Yellow" alternate="yes"
  type="text/css" href="yellow.css" ?>
<?xml-stylesheet title="Pink" alternate="yes"
  type="text/css" href="pink.css" ?>
<person>
  <fname>Arkadiusz</fname><name>Gierasimczyk</name>
  ...
</person>
```

Cascading Style Sheets – history

- Roots of stylesheet idea – 1970s:
 - translation of markup documents to (different) printer languages
- Beginning of CSS: 1994
- CSS Level 2: May 1998
- CSS 2.1: June 2011
 - restricts CSS 2 and makes it more precise
- CSS Level 3: split into modules, some of them are final recommendations, some are not

Applications of CSS

- First and major one: style for Web sites
- Separation of content and style for HTML
- (Simple) style sheets for XML
- CSS 2 and the idea of “accessibility”:
 - support for different media
 - support for alternative presentation means (e.g. voice generation)
 - enabling **reader** to override style proposed by author (reader rules)

Example stylesheet (fragment)

```
<company>
  <name>Extremely professional staff</name>
  <department id="acc">
    <name>Accountancy</name>
    <person position="expert" id="102103">
      <fname>Dawid</fname><surname>Paszkiewicz</surname>
      <phone type="office">+48223213203</phone>
      <phone type="mobile">+48501502503</phone>
      <email>paszkiewicz@example.com</email>
    </person>
    <person position="chief" id="102104">
      <fname>Monika</fname><surname>Domżałowicz</surname>
      <phone type="office">+48223213200</phone>
      <email>mdom@example.com</email>
    </person>
  </department>
  <main-office> ... </main-office>
</company>
```

Example stylesheet (fragment)

```
person {
  display: block;
  margin: 10px auto 10px 30px;
  padding: 0.75em 1em;
  width: 200px;
  border: solid 2px #002288;
  background-color: #FFFFFF;
}
person[position='chief'] {
  background-color: #DDFFDD;
}
fname, surname {
  display: inline;
  font-size: larger;
}
person[position='chief'] surname {
  font-weight: bold;
}
```

Resulting visualisation

Extremely professional staff

Accountancy

Dawid Paszkiewicz
tel. +48223213203
mob. +48501502503
paszkiewicz@example.com

Monika Domżałowicz
tel. +48223213200
mob. +48501502513
mdom@example.com

CSS selectors (representative examples)

- `surname` - element of the given name
- `fname, surname` - both elements
- `company name` - name being descendant of company
- `company > name` - name being direct child of company
- `surname + phone` - phone directly succeeding surname
- `phone:first-child` - phone being first child of its parent
- `person[position]` - person owing position attribute
- `person[position='manager']` - person with position attribute equal to manager
- `person [roles~='manager']` - person with attribute role containing word manager (attribute as space-separated list)
- `ol.staff` - equivalent to `ol[class~='staff']` (HTML only)
- `person#k12` - person with ID (in DTD meaning) equal to k12

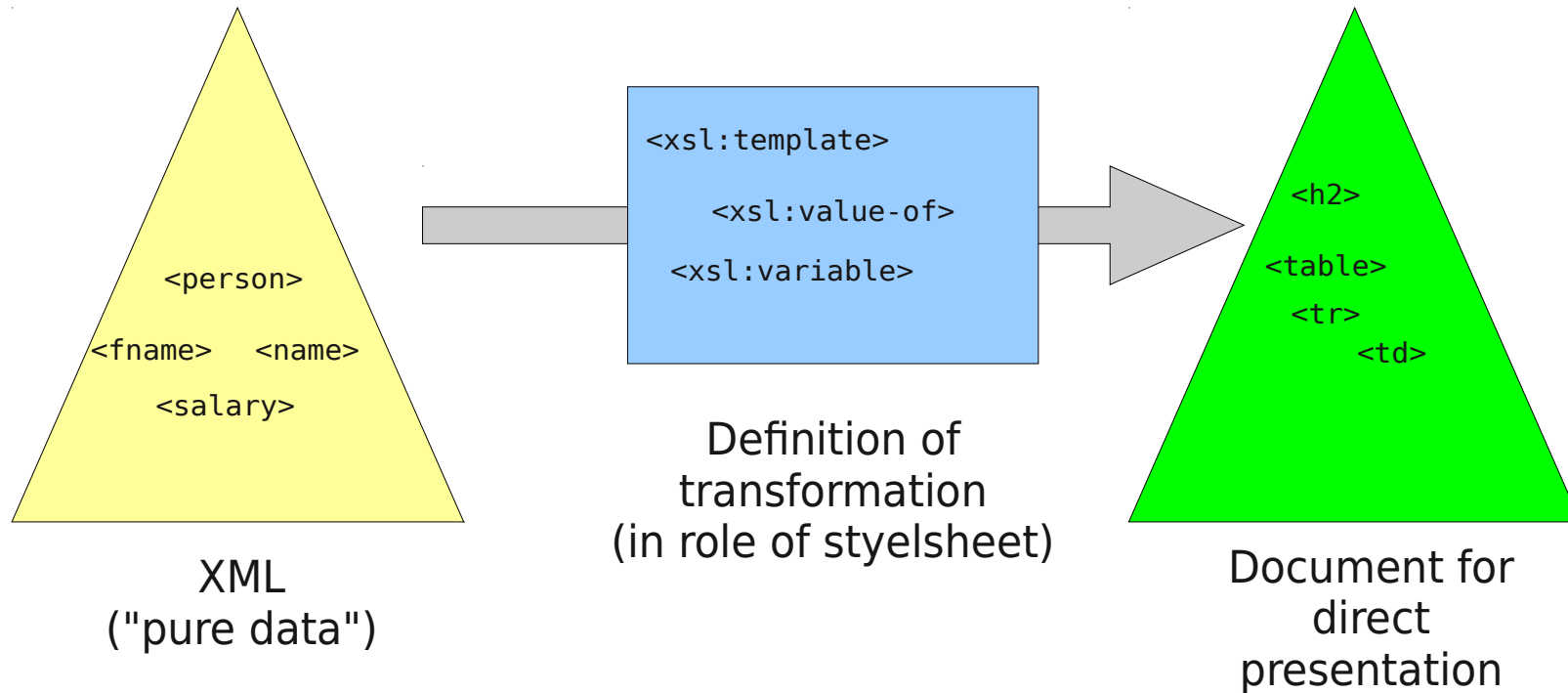
CSS capabilities and advantages

- Rich visual formatting features
- Selecting elements by
 - name
 - location in document tree
 - attribute existence
 - attribute values
- Good support
 - internet browsers
 - authoring tools
- Easy to write simple stylesheets :)

CSS shortcomings

- Only visualisation, not translation to different formats
- Selectors relatively weak. Conditions not expressible in CSS:
 - checking content of element, e.g.:
 - element **A** that contains element **B**
 - element **A** that contains text **abc**
 - logical composition of many conditions (available to some extent, but inconvenient)
 - value comparison (e.g. show negative amounts in red)
- Structure of blocks directly based on structure of source elts
 - reordering of elements hard (and not possible in general way)
 - not possible to show one element several times on page
- No data processing. Not available for example:
 - number calculations (summing etc.)
 - operations on text (shortening, regexp matching, etc.)

Presentation by transformation



Extensible Stylesheet Language (XSL)

- Defined in W3C recommendations (v1.0 in 1999 and 2001):
 - **XSL** - general framework and **XSL Formatting Objects**
 - **XSLT** - language for defining XML transformations
 - **XPath** - expression language, including paths for document fragments addressing
- Original approach:
 - Transformation definition (XSLT), in the role of stylesheet, specifies how a source document is translated into FO document.
 - Presentation of result FO is specified by XSL-FO standard and available through a rendering engine.
- Practice:
 - HTML result format used more often (although FO also used)
 - XSLT and XPath also used for purposes other than presentation

Transformation to HTML - example (1)

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" encoding="utf-8" />
  <xsl:template match="/">
    <html>
      <head>
        <title>
          Employees of <xsl:value-of select="/company/name"/>
        </title>
        <style type="text/css">
          body { background-color: #FFFFDD; ... }
          div.person { margin: 10px auto 10px 30px; ... }
          ...
        </style>
      </head>
      <body>
        <xsl:apply-templates />
      </body>
    </html>
  </xsl:template>
```

Transformation to HTML - example (2)

```
<xsl:template match="person">
  <xsl:variable name="mgr">
    <xsl:if test="@position='manager'">manager</xsl:if>
  </xsl:variable>
  <div class="person {$mgr}">
    <div class="name">
      <xsl:apply-templates select="fname" />
      <xsl:text> </xsl:text>
      <xsl:apply-templates select="surname" />
    </div>
    <div class="phone">
      <xsl:apply-templates select="phone" />
    </div>
    <div class="email">
      <xsl:apply-templates select="email" />
    </div>
  </div>
</xsl:template>
```

Transformation to HTML - example (3)

```
<xsl:template match="company/name">
  <h1>
    <xsl:apply-templates />
  </h1>
</xsl:template>
<xsl:template match="department/name">
  <h2>
    <xsl:apply-templates />
  </h2>
</xsl:template>
<xsl:template match="phone">
  <xsl:apply-templates />
  <xsl:if test="position() != last()">
    <xsl:text> </xsl:text>
  </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

Resulting HTML code (fragments)

```
<html>
<head>
  <META http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Employees of Extremely professional company</title>
  <style type="text/css">
    body {
      ...
    }
  </style>
</head>
<body>
  <h1>Extremely professional company</h1>
  <h2>Accountancy</h2>
  <div class="person ">
    <div class="name">Dawid Paszkiewicz</div>
    <div class="phone">+48223213203 +48501502503</div>
    <div class="email">paszkiewicz@example.com</div>
  </div>
  <div class="person manager">
    <div class="name">Monika Domżałowicz</div>
    <div class="phone">+48223213200 +48501502513</div>
    <div class="email">mdom@example.com</div>
  </div>
  ...

```

HTML – resulting formatting

Extremely professional company

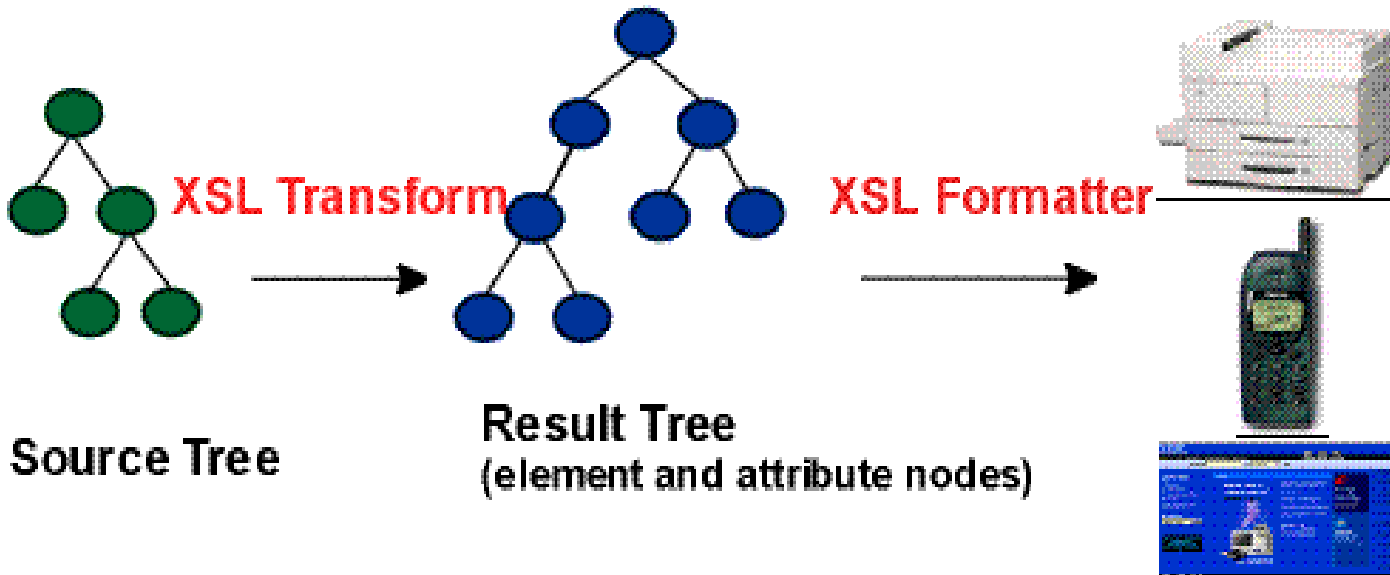
Accountancy

Dawid Paszkiewicz
+48223213203 +48501502503
paszkiewicz@example.com

Monika Domżałowicz
+48223213200 +48501502513
mdom@example.com

Marek Kącki
+48223213212 +48501502524
kacki@example.com

Original idea of XSL



Result XML tree is the result of XSLT processing.

Source: *Extensible Stylesheet Language (XSL) Version 1.0*,
W3C Recommendation 15 October 2001
(<http://www.w3.org/TR/xsl/>)

Transformation to XSL-FO - example (1)

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:output method="xml" encoding="utf-8" />
<xsl:template match="/">
  <fo:root>
    <fo:layout-master-set>
      <fo:simple-page-master master-name="A4"
        page-width="210mm" page-height="297mm" margin="1cm">
        <fo:region-body margin="16pt 0" />
        <fo:region-before extent="16pt" />
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="A4">
      <fo:static-content flow-name="xsl-region-before">
        <fo:block>
          Employees of <xsl:value-of select="/company/name"/>.
        </fo:block>
      </fo:static-content>
      <fo:flow flow-name="xsl-region-body">
        <xsl:apply-templates />
      </fo:flow></fo:page-sequence>
    </fo:root></xsl:template>
```

Transformation to XSL-FO - example (2)

```
<xsl:template match="person">
  <fo:block font-family="Verdana, sans-serif"
    space-before.minimum="12pt" padding="0.5em"
    border-width="1.5pt" border-style="solid"
    border-color="#664400" background-color="#FFFFCC">
    <fo:block font-size="14pt">
      <xsl:apply-templates select="fname" />
      <xsl:text> </xsl:text>
      <xsl:apply-templates select="surname" />
    </fo:block>
    <fo:block margin-top="0.5em">
      <xsl:apply-templates select="phone" />
    </fo:block>
    <fo:block margin-top="0.5em">
      <xsl:apply-templates select="email" />
    </fo:block>
  </fo:block>
</xsl:template>
<xsl:template match="person[@position='manager']/surname">
  <fo:inline font-weight="bold">
    <xsl:apply-templates />
  </fo:inline>
</xsl:template>
```


Transformation to XSL-FO - example (3)

```
<xsl:template match="phone">
  <fo:block>
    <xsl:choose>
      <xsl:when test="@type='mobile'">mob. </xsl:when>
      <xsl:otherwise>tel. </xsl:otherwise>
    </xsl:choose>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>
<xsl:template match="email">
  <fo:block font-style="italic">
    <xsl:apply-templates />
  </fo:block>
</xsl:template>
<xsl:template match="department/name">
  <fo:block font-size="16pt" font-weight="bold" font-style="italic"
    text-align="left" margin-bottom="6pt">
    <xsl:apply-templates />
  </fo:block>
</xsl:template>
<!-- Some more templates... -->
```

Resulting XSL-FO code (fragments)

```
<?xml version="1.0" encoding="utf-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="A4" page-width="210mm" ...>
    ...</fo:layout-master-set>
  <fo:page-sequence master-reference="A4">
    <fo:static-content flow-name="xsl-region-before">
      <fo:block>Employees of Extremely professional
        company.</fo:block>
    </fo:static-content>
    <fo:flow flow-name="xsl-region-body">
      <fo:block ...> ...
        <fo:block space-before.minimum="12pt" padding="0.5em" ...>
          <fo:block font-size="14pt"> Monika
            <fo:inline font-weight="bold">Domżałowicz</fo:inline>
          </fo:block>
          <fo:block margin-top="0.5em">
            <fo:block>tel. +48223213200</fo:block>
            <fo:block>mob. +48501502513</fo:block>
          </fo:block>
          <fo:block margin-top="0.5em">
            <fo:block font-style="italic">mdom@example.com</fo:block>
          </fo:block></fo:block>...
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

XSL-FO – resulting formatting

Employees of Extremely professional company.

Extremely professional company

Accountancy

Dawid Paszkiewicz

tel. +48223213203

mob. +48501502503

paszkiewicz@example.com

Monika **Domżałowicz**

tel. +48223213200

mob. +48501502513

mdom@example.com

XSL-FO – basic facts

- Presentation-oriented XML application
- Elements for different kinds of visual objects (**block**, **inline**, **table**, and so on)
- Attributes for formatting, based on CSS properties
- Especially useful for printed publications
- Focused on paged media type:
 - master pages (templates), page areas (header, footer, etc.)
 - automatic text flow and repeated (“static”) content
 - Practice: intermediate format in XML → XSL-FO → PDF transformation
 - Not supported by web browsers
- Designed as part of XSL framework
 - result of XSLT transformation
 - not intended to be used standalone

Basic structure of XSL-FO document

- Two main parts: declarations and actual content

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="my-page">
      <fo:region-body />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="my-page">
    <fo:flow flow-name="xsl-region-body">
      <fo:block>Hello World!</fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Formatting objects

XSL-FO elements relate to resulting formatting objects.

- Block level
 - `block`
 - `list-block`, `list-item`, `list-item-label`
 - `table`, `table-row`, `table-cell`, ...
- Inline level
 - `inline`, `character`
 - `external-graphics`
- Special features
 - `basic-link`, `bookmark`, `marker`
 - `footnote`
 - `page-number`

List - example

```
<fo:list-block>
  <fo:list-item>
    <fo:list-item-label>
      <fo:block>First name: </fo:block>
    </fo:list-item-label>
    <fo:list-item-body>
      <fo:block margin-left="15em">Dawid</fo:block>
    </fo:list-item-body>
  </fo:list-item>
  <fo:list-item>
    <fo:list-item-label>
      <fo:block>Surname: </fo:block>
    </fo:list-item-label>
    <fo:list-item-body>
      <fo:block margin-left="15em">Paszkiewicz</fo:block>
    </fo:list-item-body>
  </fo:list-item>
</fo:list-block>
```

Table - example

```
<fo:table border="solid 2pt black">
  <fo:table-header>
    <fo:table-row>
      <fo:table-cell><fo:block font-weight="bold">Surname
                          </fo:block></fo:table-cell>
      <fo:table-cell><fo:block font-weight="bold">First name
                          </fo:block></fo:table-cell>
      ...
    </fo:table-row>
  </fo:table-header>
  <fo:table-body>
    <fo:table-row>
      <fo:table-cell><fo:block>Paszkiwicz</fo:block></fo:table-cell>
      <fo:table-cell><fo:block>Dawid</fo:block></fo:table-cell>
      ...
    </fo:table-row>
```


Formatting properties

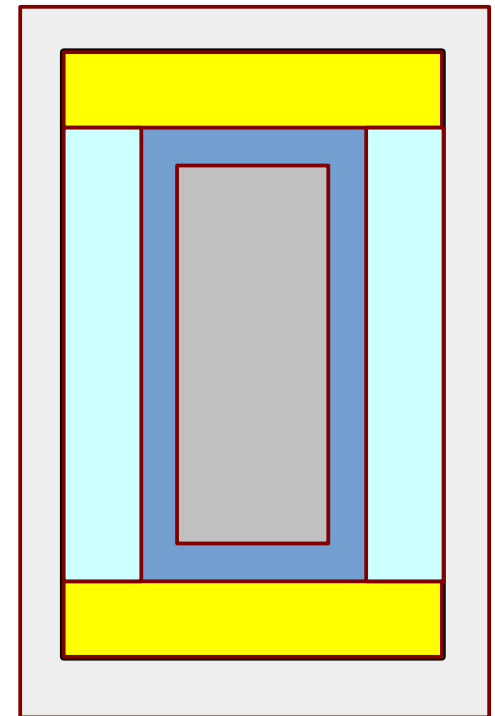
Most of XSL-FO attributes relate to style properties analogue to CSS properties.

- `margin`, `padding`, `border-style`
- `background-color`, `background-image`
- `font-family`, `font-weight`, `font-style`, `font-size`
- `text-align`, `text-align-last`, `text-indent`, `start-indent`, `end-indent`, `wrap-option`,
- `break-before`
- and much more (almost 300 properties in XSL 1.1)

“Page master” – page template

- Single page layout
- A document may be split in many such pages
- One body region (or more in XSL 1.1)
- Four predefined (but optional to use) edge regions

```
<fo:simple-page-master master-name="A4"  
  page-width="297mm"   page-height="210mm"  
  margin="1cm">  
  <fo:region-body      margin="3cm" />  
  <fo:region-before   extent="2cm" />  
  <fo:region-after    extent="2cm" />  
  <fo:region-start    extent="2cm" />  
  <fo:region-end      extent="2cm" />  
</fo:simple-page-master>
```



Distributing content to pages

- **page-sequence** - results in a number of pages
- **flow** - content split into pages
- **static-content** - content repeated on all pages
- **flow-name** - page region reference

```
<fo:page-sequence master-reference="A4">
  <fo:static-content flow-name="xsl-region-before">
    Employees of <xsl:value-of select="company/name" />
  </fo:static-content>

  <fo:flow flow-name="xsl-region-body">
    <xsl:apply-templates />
  </fo:flow>
</fo:page-sequence>
```

Page sequence master

- Using different page layouts within one page-sequence
- Simple page masters referred to be used in order (repetitions available)

```
<fo:layout-master-set>
  <fo:simple-page-master master-name="first">...</fo:simple...>
  ...
  <fo:page-sequence-master master-name="seq_master">
    <fo:single-page-master-reference master-reference="first"/>
    <fo:repeatable-page-master-reference
      master-reference="starting" maximum-repeats="3"/>
    <fo:repeatable-page-master-reference
      master-reference="default"/>
  </fo:page-sequence-master>
</fo:layout-master-set>

<fo:page-sequence master-reference="seq_master">
  <fo:flow flow-name="xsl-region-body">...
```

Alternative page references

- Choosing page depending on conditions
 - odd/even, blank?, first?

```
<fo:page-sequence-master master-name="rich_master">
  <fo:repeatable-page-master-alternatives>
    <fo:conditional-page-master-reference
      master-reference="first" page-position="first"/>
    <fo:conditional-page-master-reference
      master-reference="right" odd-or-even="odd"/>
    <fo:conditional-page-master-reference
      master-reference="left" odd-or-even="even"/>
  </fo:repeatable-page-master-alternatives>
</fo:page-sequence-master>
```

Multiple flows

- Added in XSL 1.1, not available in Apache FOP

```
<fo:layout-master-set>
  <fo:simple-page-master master-name="two-bodies"
    page-width="210mm" page-height="297mm" margin="10mm">
    <fo:region-body margin="27mm 0 100mm 0" region-name="top"/>
    <fo:region-body margin="177mm 0 0 0" region-name="down"/>
    <fo:region-before extent="27mm" />
  </fo:simple-page-master>
</fo:layout-master-set>

<fo:page-sequence master-reference="two-bodies">
  <fo:static-content flow-name="xsl-region-before">
    <fo:block>Header</fo:block>
  </fo:static-content>
  <fo:flow flow-name="top">
    <xsl:apply-templates select="main-content"/>
  </fo:flow>
  <fo:flow flow-name="down">
    <xsl:apply-templates select="bottom-content"/>
  </fo:flow>
</fo:page-sequence>
```

Custom flow maps - by example

- Here: two text flows merged together and placed into two page regions

```
<fo:flow-map flow-map-name="E4">
  <fo:flow-assignment>
    <fo:flow-source-list>
      <fo:flow-name-specifier flow-name-reference="A"/>
      <fo:flow-name-specifier flow-name-reference="B"/>
    </fo:flow-source-list>
    <fo:flow-target-list>
      <fo:region-name-specifier
        region-name-reference="R1"/>
      <fo:region-name-specifier
        region-name-reference="R2"/>
    </fo:flow-target-list>
  </fo:flow-assignment>
</fo:flow-map>
```

Markers

- Typical application - "running header"

In flow (main content)

```
<xsl:template match="section">
  <fo:block ...>
    <fo:marker marker-class-name="sec-name">
      <xsl:value-of select="title"/>
    </fo:marker>
  </fo:block>
  ...
</xsl:template>
```

In static content (header)

```
<fo:block>
  Page <fo:page-number />
  Section <fo:retrieve-marker retrieve-class-name="sec-name"/>
</fo:block>
```


XSL-FO – discussion

- Main XSL-FO advantages
 - consistent with XSL idea
 - easy and direct way to obtain printout (e.g. PDF) from XML data
 - general advantages of stylesheets over “hard-coded” formatting
 - automatised wrt. manual formatting in text editors
- Main XSL-FO disadvantages
 - too complex for simple needs (see e.g. lists)
 - too limited for advanced needs
 - lack of pagination feedback, cannot say “if these two elements occur on the same page then...”
 - hard to format particular elements in a very special way (this is the general drawback of automated stylesheets compared to manual formatting)

Desktop Publishing (DTP)

- Production of high-quality text&graphics material to be printed (main focus) or published in different ways
 - Examples: press, marketing folders, user manuals
- Existing approaches to work:
 - manual preparation of all materials in specialised tools
 - semi-automated workflow, e.g.:
 - manual preparation of templates (often by example documents)
 - creation of actual documents by filling the template with varying content
 - frequent need of manual corrections after the template is filled with actual data
 - fully automated production – rarely applied

DTP-related terms

- **Template** – a document baselining the structure, shape (page size, margins) and format (available styles, etc.) of documents
 - Templates are often used to produce series of documents varying in their content, but sharing a common structure and style.
- **Page master** – a template of a page, fixing its orientation, size, margins, and setting available regions on the page
 - Depending on technology, page master may also set the content of static page regions, usually header and footer.
 - A document may use several page masters, e.g. different masters for odd and even pages, and a separate one for the title page.
- **Flow** – a sequence of content distributed on document pages
 - In advanced DTP, a document may have more than one flows. It is possible to have concurrent flows on the same page.

Why XML in DTP workflow?

- Typical DTP tool formats: proprietary, closed, requiring commercial products to access documents
- Many tools and technologies making use of XML and extending particular tool functionality:
 - general technologies supporting XML (XSL, XQuery)
 - custom applications based on programming libraries
 - CMS, report generators, Web Services, etc.
 - specialised tools for particular XML applications (DITA, DocBook, RSS, MathML, SVG, ...)
- Communication between a “DTP tool world” and the external world

DTP and XML – different approaches

- XML as additional format required from time to time
 - *Save as XML...* available (at least in tools presented today)
- XML as central format in workflow process
 - *structured application* developed
 - content stored in XML files
 - DTP tools used as editors and formatting engines
 - additional tools may consume XML (CMS, for instance)
- DTP tool for formatting purposes only
 - XML created (manually or automatically) independently
 - DTP tools used to open and “print” document (e.g. by exporting PDF)
 - DTP tools and their templates play role analogous to stylesheets
 - manual enhancements available in special cases
 - which would not be possible using generic stylesheets, e.g. XSL-FO

As an example – Adobe FrameMaker

- No intention to advertise a particular tool – just an example of something which is:
 - really used in DTP industry
 - focused on (rather) large structural documents
 - using XML.

Adobe FrameMaker

- Word processor / desktop publishing tool
 - One of first so advanced tools
 - Acquired by Adobe in 1995
- Especially popular for:
 - complex documents, where structure important
 - large documents, e.g. technical documentation
- Two kinds of documents (and 2 ways of authoring):
 - **unstructured** - flat, paragraph-based structure, similar to styles in popular word processors
 - **structured** - tree-like structure, based on SGML and XML

FrameMaker augments a structural approach to the content with a WYSIWYG editor convenience.

Structured documents in FM

- Structured application
 - FM concept analogous to XML application in XML world
- FM manages a set of registered structured applications
- XML documents opened / saved directly:
 - Template and formatting rules from EDD define the formatting.
 - When an XML file is opened, the XML content fills the template and formatting rules define the style applied to the content...
 - Manual formatting available in FM, but lost when document saved as XML.

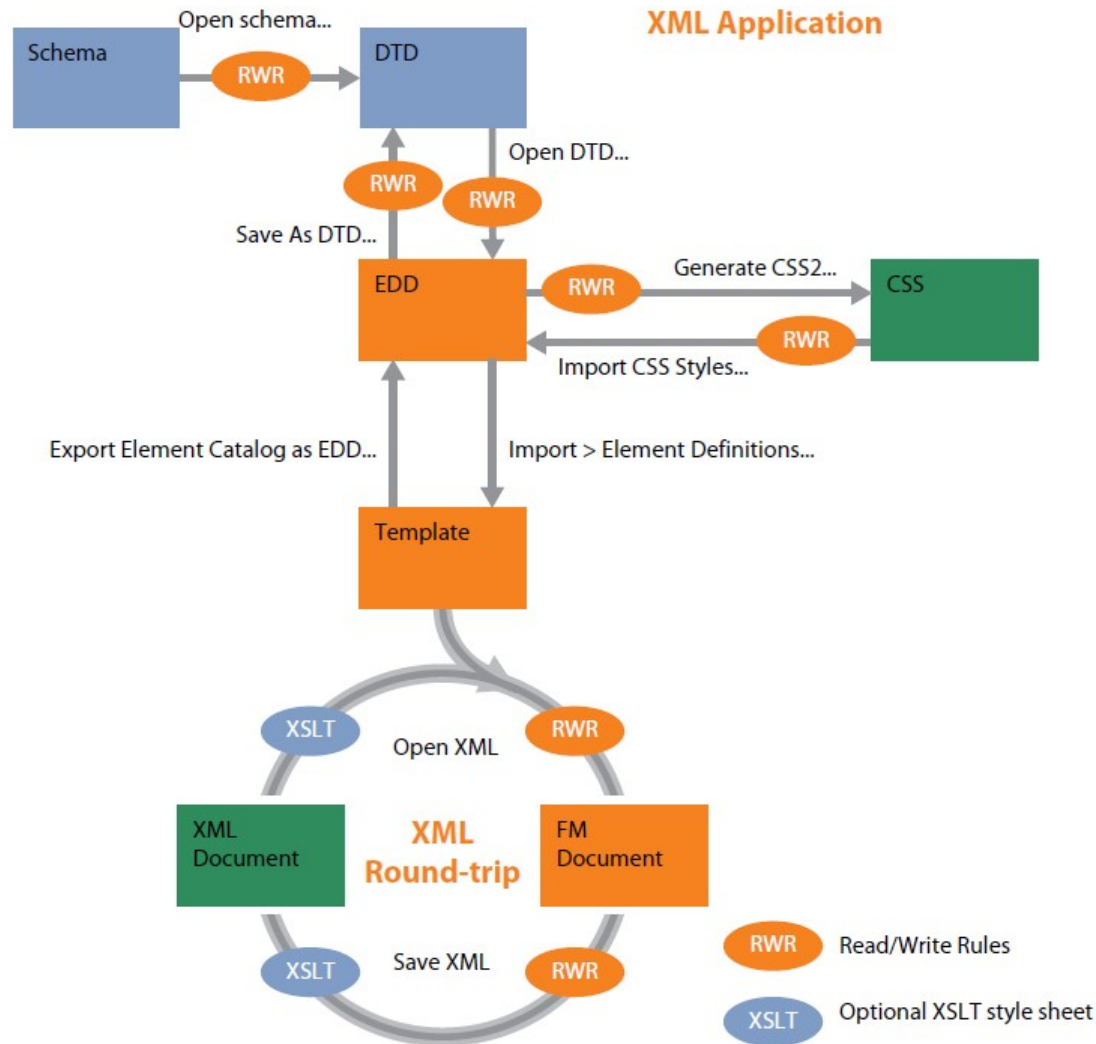
Structured application

- **EDD** – Element Definition Document (or Elements Catalogue)
 - document structure definition (elements, attributes)
 - formatting and other rules
- **structured template** – FM document
 - pagination, layout, header and footer, ...
 - styles (“paragraph/character format tags”), variables, markers, cross-reference formats, ...
 - Elements Catalogue imported from EDD

Optional components:

- **DTD** – may also be generated from EDD
- **Read/write rules** – extra translations between XML and FM
- **XSLT pre- and post-processing**
- **API client** – custom executable application

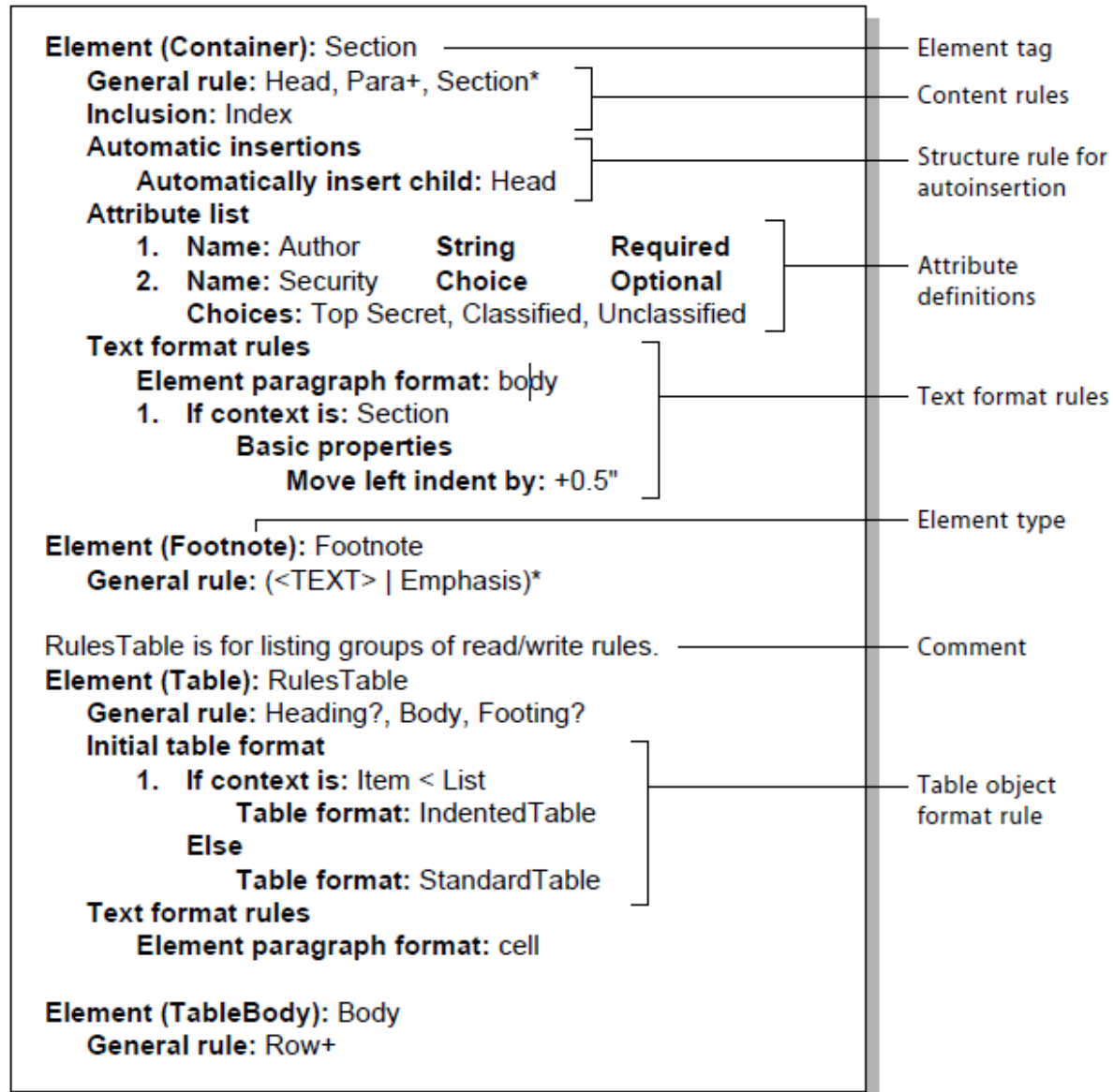
Structured application dependencies



Element Definition Document

- FM document defining other documents structure
- EDD role corresponding to (in general XML applications):
 - DTD or XML Schema - structure definition
 - CSS or XSL (to some extent) - formatting rules
- Structure definition
 - available elements, their type and acceptable content
 - attributes, their type and optionality
- Particular elements marked as FM special objects (tables and table components, variables, markers, cross-references, ...)
- Rules for elements:
 - formatting
 - initial value or structure
 - prefix and suffix

Element definition examples (EDD shown in document view)



Content model (General rule)

- Expression built from element names, <TEXT> token, parentheses, and:
 - grouping symbols (between element names or () groups)
 - , - sequence of subelements
 - & - subelements in any order
 - | - choice
 - occurrence indicators (after element name or () group):
 - ? - optional element (0-1 occurrence)
 - * - any number of occurrence (0-unbounded)
 - + - at least one occurrence (1-unbounded)
 - no indicator - exactly one occurrence
- Examples:
 - `imię+, nazwisko`
 - `Title, Abstract?, Section*`

Kinds of elements (examples)

- **Container**
 - element with no special meaning
 - may contain elements or text (or both → *mixed model*)
- **CrossReference** – FM cross-reference
- **Footnote** – FM footnote
- **Equation, Graphic** – anchored objects;
XML would contain references to external entities
- **Marker** – FM marker
- **SystemVariable** – FM system variable reference
- **Table, TableBody, TableHeading, TableRow, TableCell**
– table components

EDD and DTD – similarities

- Document structure definition
- **Container** elements
- Content model specification (| , * ? +)
- Optional and required attributes
- **Unique ID, ID Reference** – ID, IDREF in DTD

EDD and DTD – differences

EDD

- FrameMaker-special element kinds (tables, variables, etc.)
- Numeric attribute types
- Multi-value attributes
- & – elements in any order
- Formatting rules
- No means for structure modularisation
 - style modularisation available through format change lists

DTD

- General-purpose elements (like EDD **Container**)
- No numeric types (for XML)
- Space-separated NMTOKENS and IDREFS
- Only choice and sequence
- No formatting specification
- Parameter entities as means for DTD modularisation

EDD and XML Schema

EDD

- FM-tied (special element kinds, formatting)
- No constraints for simple values, except lists of choice for attributes
- General ID/IDREF mechanism
- No means for structure modularisation
- Format specific for FM

XML Schema

- General-purpose technology (like DTD)
- Simple types and precise control of simple values (text, numbers, etc.)
- Advanced key/keyref mechanism
- Modularisation through types, type inheritance, groups
- Understandable and usable outside FM world (e.g. for WebServices)

Formatting rules

- Appearance of particular elements described in EDD
- In element definition (e.g. **Container**) rules grouped by scope of effect:
 - **TextFormatRules** – formatting of whole element, inherited by descendants
 - **FirstParagraphRules, LastParagraphRules** – formatting of first / last paragraph only
 - **PrefixRules, SuffixRules** – content generated in front / at end of element and its formatting
- Some more features analogous to CSS selectors:
 - context rules
 - level rules
- We omit the rest of details here...

Format rules - example

Element (Container): Head

General rule: <TEXT>

Text formal rules

1. In all contexts

Default font properties

Weight: Bold

Size: 14pt

Numbering properties

Autonumber format: <n>.<n+>\t

Context rules - example

Text format rules

1. If context is: List [Type = "Bulleted"]

 Numbering properties

 Autonumber format: \b\t

 Character format: bulletsymbol

Else, if context is: List [Type = "Numbered"]

 1.1 If context is: {first}

 Numbering properties

 Autonumber format: <n=1>\t

 Else

 Numbering properties

 Autonumber format: <n+>\t