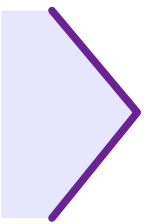
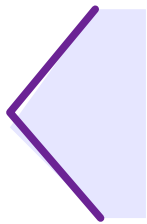


# Presentation of XML – commercial approach (XML in DTP)

Patryk Czarnik

XML and Applications 2013/2014  
Week 11 – 16.12.2013



- Uwaga, nie trzeba uczyć się wszystkich szczegółów technicznych do egzaminu.

# 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 tamplate 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 its templates play role analogous to stylesheets
  - manual enhancements available in special cases
    - which would not be possible using generic stylesheets, e.g. XSL-FO

# Tools mentioned today

- Adobe FrameMaker
  - especially useful for large and complex text documents
  - advanced support for XML and structured documents
  - constructs analogous to DTD and stylesheets
- Adobe InDesign
  - especially useful for documents that have to look perfectly
  - basic support for XML by means of filling a template with XML content

# Adobe FrameMaker

- Word processor / desktop publishing tool
  - One of first that 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.



# Basic XML features

- For any FM document  
File > Save As XML...
- Unstructured document
  - XML structure based on styles and FM objects (tables etc.)
- Structured document
  - XML structure directly reflecting document structure
- Structured document within registered structured application
  - *Read/write rules* and XSLT postprocessing may additionally affect resulting XML.

# Structured documents in FM

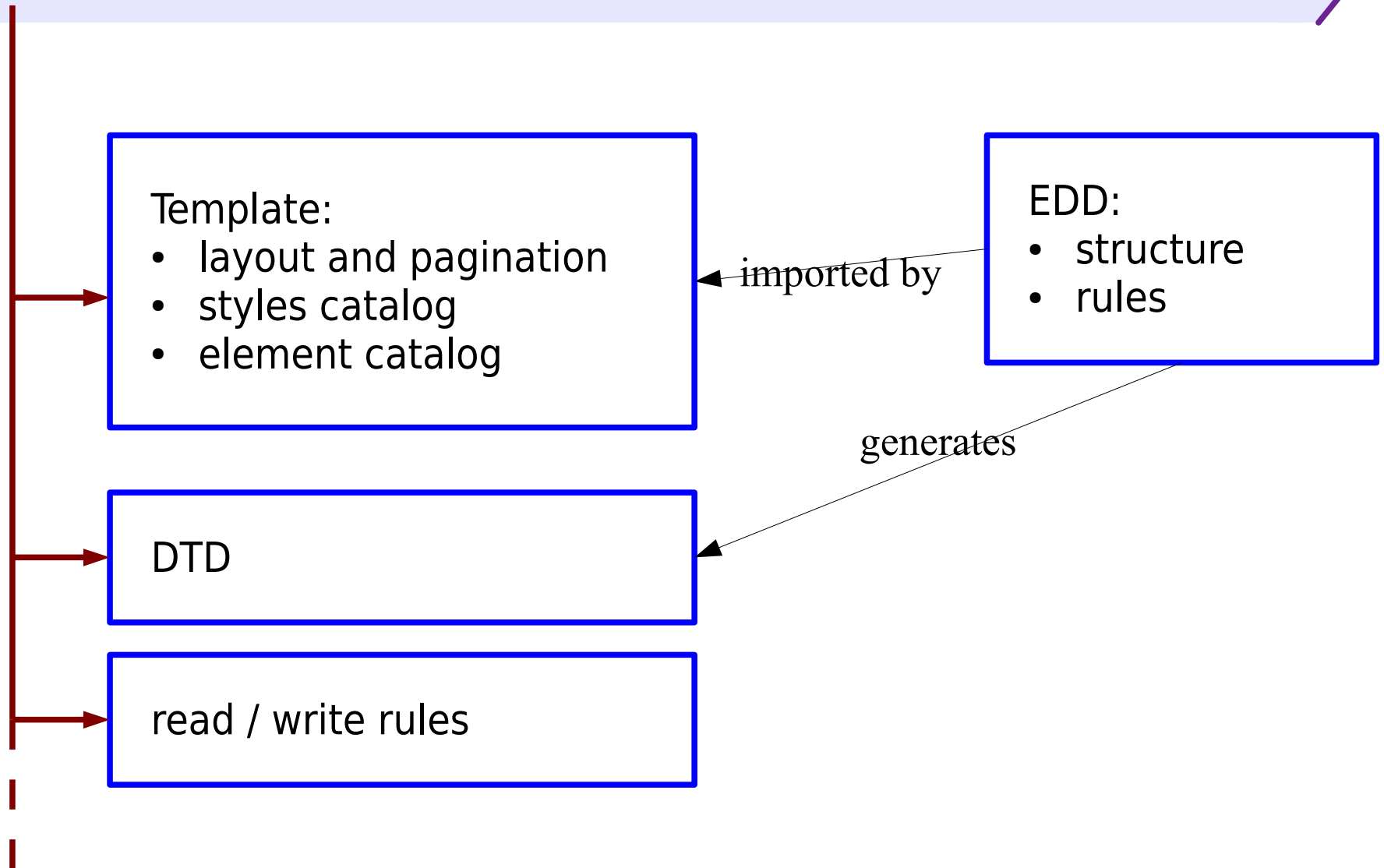
- Structured application
  - FM concept analogous to XML application in XML world
- FM manages a set of registered structured applications
- XML documents opened / saved directly
  - template and formatting rules from EDD define the formatting
  - manual formatting available in FM, but lost when document saved as XML

# Structured application

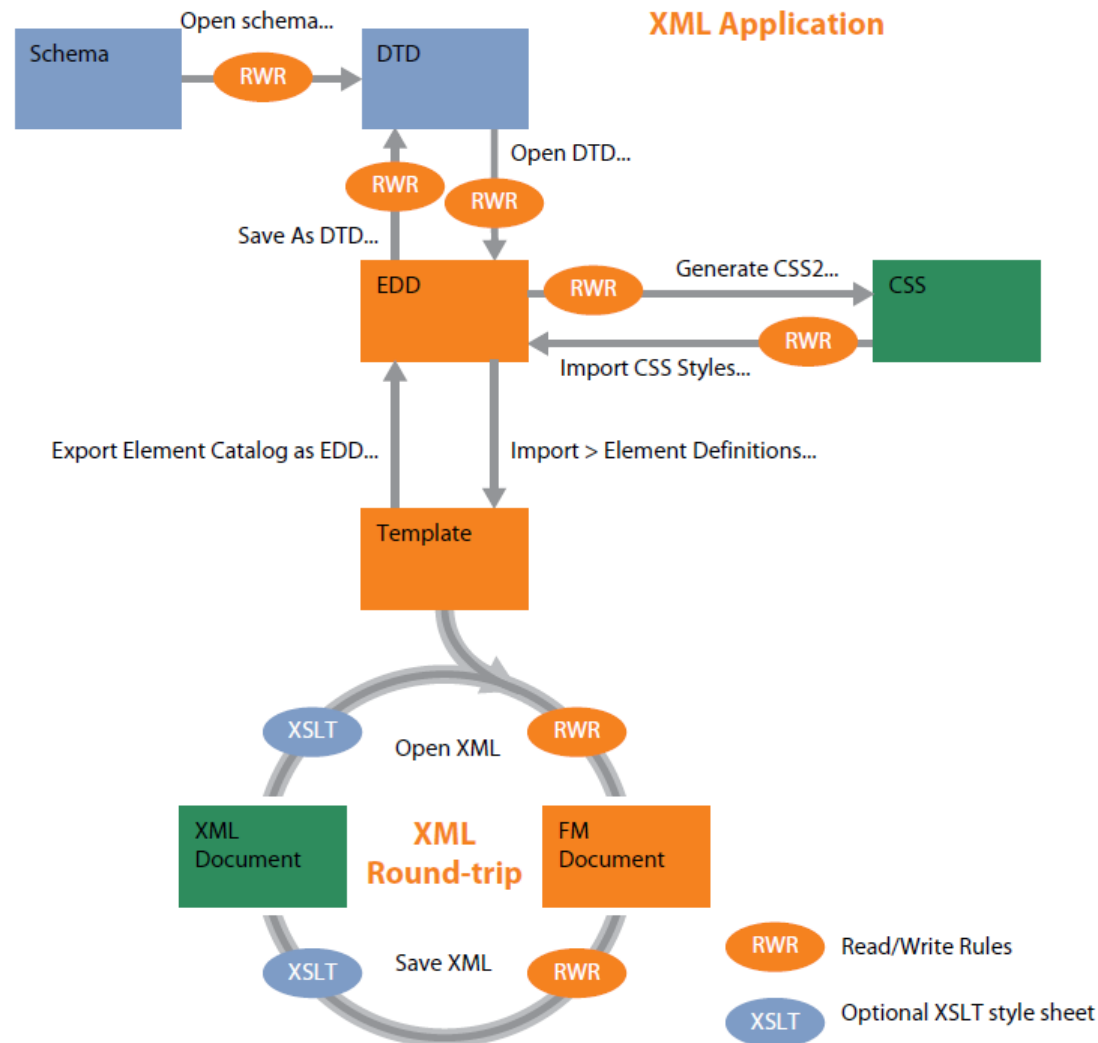
- **EDD** – Element Definition Document (or Elements Catalogue)
  - document structure definition (elements, attributes)
  - formatting and other rules

! not included in s.a. definition directly, rather through template
- **DTD** – may be generated from EDD
- **structured template** – FM document
  - pagination, layout, header and footer, ...
  - styles (“paragraph/character format tags”), variables, markers, cross-reference formats, ...
  - Elements Catalogue imported from EDD
- **Read/write rules** – extra translations between XML and FM
- **XSLT pre- and post-processing**
- **API client** – custom executable application

# Structured application structure :)



# Structured application dependencies



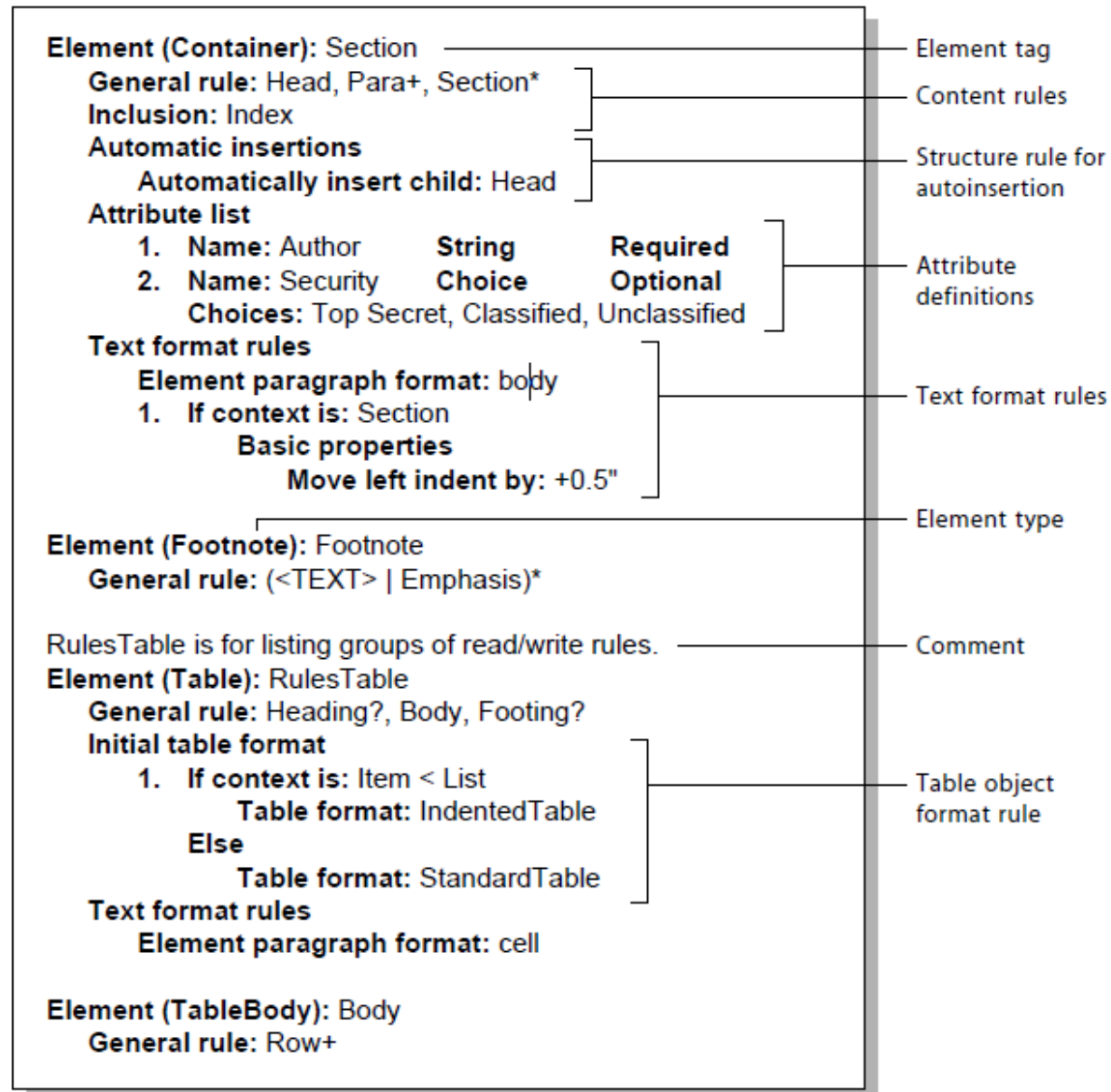
# Element Definition Document

- 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

# EDD based on existing XML application

- Options for EDD creation:
  - from scratch
  - based on existing DTD
  - based on existing XML Schema
- If based on existing structure, some details to be added:
  - formatting rules (DTD or XML Schema do not contain such)
  - relation to FM special objects (tables, variables, etc.)
- Formatting rules may be created based on CSS.
  - Though many details are not reflected and have to be recovered manually.

# 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

- **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
- **Rubi, RubiGroup** – Asian alphabets support

## Kinds of elements (cntnd.)

- **Table** – FM table main element
- **TableTitle, TableBody, TableHeading, TableFooting, TableRow** – table components
- **TableCell** – table cell; may contain text and other elements (like **Container**) but no table or table component

# Types of attributes

- **Choice** – one of given values
- **String** – any text
- **Integer** – integer number
- **Real** – floating-point number in decimal (e.g. 0.0023) or exponential (e.g. 2.3e-3) notation
- **Unique ID** – value unique within document scope
- **ID Reference** – reference to **Unique ID** value somewhere in document
- **Strings, Integers, Reals, Unique IDs, ID References**  
– multi-value attributes

## Document structure specification – what more?

- **<ANY>** or **<EMPTY>** as content model (General rule)
- **ValidHighestLevel** – element may be document root
- **AutoInsertions, InsertChild, InsertNestedChild** – automatic insertion of subelements
- **InitialStructurePattern** – initial content (on structure level) of table

## 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

# Format change list – example

## **Format change list: Code**

Basic properties

Tab stops

Relative tab stop position: +12pt

Alignment: Left

Default font properties

Family: Courier

Pair kerning: No

Element (Container): CodeFragment

General rule: <TEXT>

Text format rules

1. In all contexts.

Text range.

**Use format change list: Code**

# Reference to document-defined style

- **ElementPgFormatTag** element
- Good practice – define all referenced formats in *structured application template*

```
Element (Container): Item
  General rule: <TEXT>
  Text format rules
    Element paragraph format: item
    1. If context is: BulletList
      Numbering properties
      Autonumber format: \b\t
    Else, if context is: NumberList
      Numbering properties
      Autonumber format: <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

# Level rules – example

Text format rules

1. Count ancestors named: Section

If level is: 1

Default font properties

Font size: 18

If level is: 2

Default font properties

Font size: 14

If level is: 3

Default font properties

Font size: 12

# Prefix / suffix rules – example

Prefix rule

1. In all contexts.

Prefix: <attribute[Label]>

Font properties

Weight: Bold

# Tables

- FM table model restrictions:
  - Table > Title?, Header?, Body, Footer? > Row+ > Cell+
- Document structure required to conform the requirements
- Arbitrary element names, obligatory element kinds
- Table cell allowed to contain text and other elements (like **Container**) but no table or table component
- *Read/write rules* must add table metadata on read (and optionally store them to XML on write)
- When table element defined in EDD, normal FM tools (here: Table creator) may be used to insert tables conveniently



# Read/write rules

- Translation between FM-internal and XML form of document
- Capabilities:
  - changing name of element or attribute
  - setting/changing attribute value
  - mapping between FM special objects (variables, markers, etc.) and XML constructs (elements, entities, processing instructions)
  - mapping between FM-specific metadata and XML elements or attributes (e.g. table metadata)
- More complex structure modifications or content processing
  - use XSLT

## R/W rules example: Storing table metadata in XML attributes

```
element "table" {  
  is fm table element "Table";  
  attribute "frame" {  
    is fm property table border ruling;  
    value "top" is fm property value top;  
    value "bottom" is fm property value bottom;  
    value "topbot" is fm property value top and bottom;  
    value "all" is fm property value all;  
    value "sides" is fm property value sides;  
    value "none" is fm property value none;  
  }  
  attribute "colsep" is fm property column ruling;  
  attribute "rowsep" is fm property row ruling;  
  attribute "numcols" is fm property columns;  
  attribute "colwidths" is fm property column widths;  
}
```

# Variables and markers

- Variables in FM
  - enable automatic insertion of varying texts into the main text flow or into the background text (header/footer)
  - predefined set of system variables (page number, etc.)
  - user-defined variables (e.g. document title)
- Markers
  - parts of document can be marked in a special way
  - and then referred from a cross-reference or the header (to make “running header”)
- In structured applications they can be represented as XML documents, if appropriately declared in EDD
  - otherwise they are written in XML as processing instructions

# Cross references

- FM cross-references represented by elements of kind **CrossReference**
- Structural cross-references – based on **Unique Id** and **IdReference** attribute types (mapped to ID/IDREF in DTD); structure requirements:
  - **IdReference** declared in reference element
  - **Unique Id** declared in referenced element
- Cross-reference format defined in actual FM document
  - good practice: in *structured template*
- Inserting structural references very easy in FM:
  - choosing elements capable of having **Unique Id**
  - ID automatically generated, if not existed so far
  - IDREF inserted behind scenes

# Conversions

Both the functionalities make use of “conversion tables”

- Migrating unstructured documents to structure
- Exporting documents to HTML

# Conversion rule examples

- Simple (paragraph style, character style):
  - P:Body para
  - C:Emphasis em
- Wrapping elements (any graphics, paragraph style, and wrapper for both):
  - G: Graphic
  - P:caption Caption
  - E:Graphic,E:Caption Figure
    - As we can see, we can apply structure to elements we have just created.
- Root:
  - RE:RootElement document

# Manual modification of conversion table

- Available on appropriate *reference page* (after first usage of HTML)

FrameMaker Source Item	XML Item		Include	Comments
	Element	New Web Page?	Auto#	
C:EquationVariables	EM	N	N	
X:Heading & Page	Heading	N	N	
X:Page	Heading	N	N	
X:See Heading & Page	See Also	N	N	
X:Table All	Table All	N	N	
X:Table Number & Page	Table Number	N	N	
E:description	DD	N	N	
E:document	UNWRAP	N	N	
E:em	EM	N	N	
E:entry	DL	N	N	
E:p	P	N	N	
E:section	DIV	Y	N	
E:term	DT	N	N	
E:title	H*	N	N	
X:TermRef	See Also	N	N	

# Adobe InDesign

- Advanced desktop publishing tool
  - part of Adobe Creative Suite
- Especially helpful when:
  - text and graphics mixed together
  - advanced, non-standard page layouts used
  - precise positioning and typesetting required
  - high-quality printout planned
- Compromise between word processor and graphic design tool
- Less structure support when compared to Adobe FrameMaker or XML-based solutions
  - harder to automatize publication process
  - easier to make ad-hoc formatting enhancements



# Main XML-related features of InDesign

- Parts of document annotated with XML-based structure
- Structure-oriented tools and features:
  - **Tags** panel and **Structure** pane
  - tag markers visible in **Story Editor** and layout view
- Tagging unstructured content
  - manual
  - automatic
  - based on styles
- Exporting tagged content as XML
- Importing XML, and then:
  - manually distributing XML fragments among text frames
  - automatic layout of imported content if placeholders were prepared
- XML import options

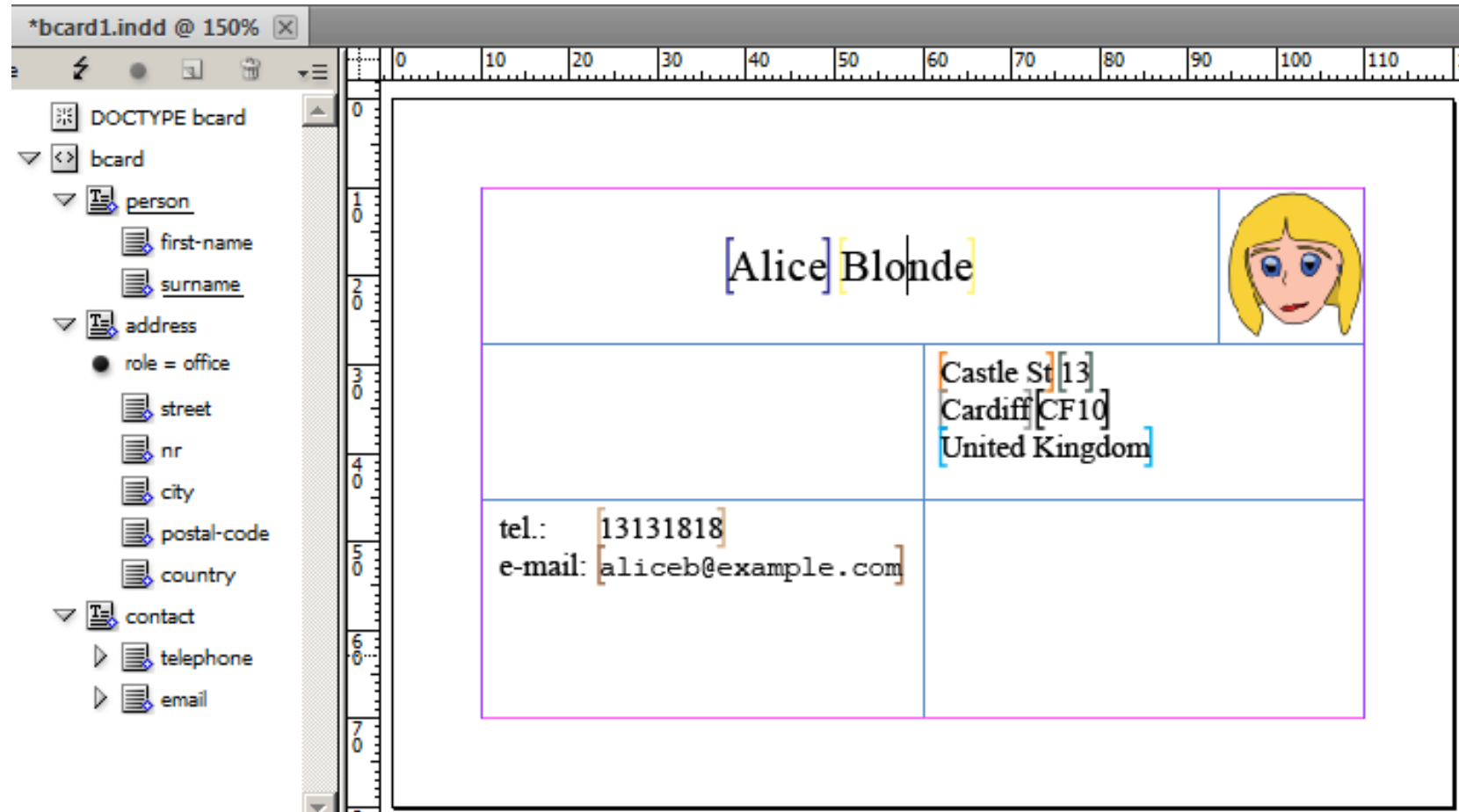
# Main XML-related features of InDesign

- Relating InDesign styles and XML tags:
  - applying styles to tags automatically
  - tagging content based on styles
- InDesign attributes in XML documents
- Script-based XML rules
- XSLT pre- and post-processing
- Saving document in concrete XML-based formats:
  - InDesign Markup Language (IDML)
  - XHTML
  - EPUB

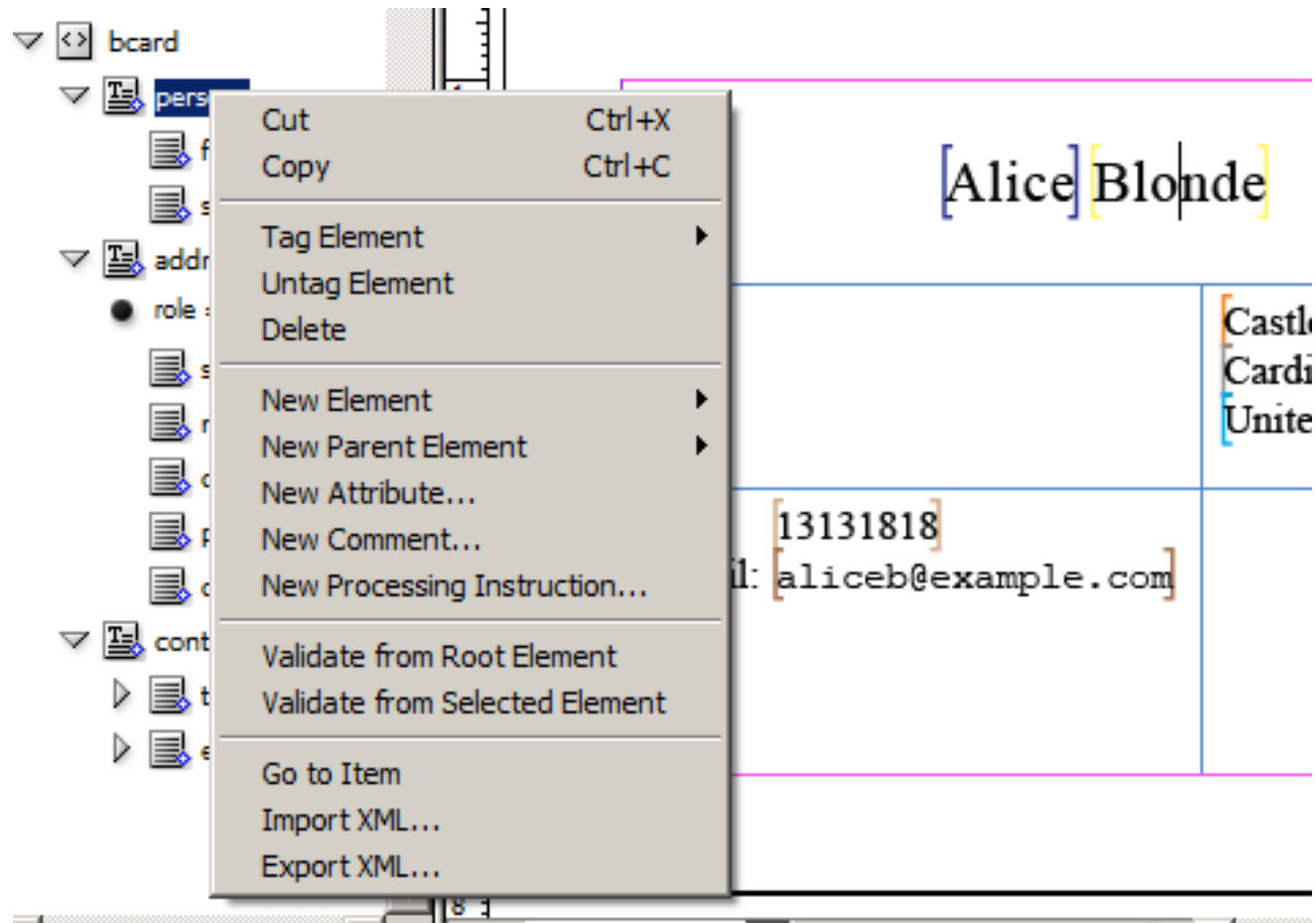
# InDesign and XML – what for?

- XML as interface language between various publishing tools
- Exporting content already existing in InDesign documents for external, structure-intensive processing
- Importing XML into InDesign documents; XML as main content:
  - InDesign template as formatting skin for structured XML content coming from external sources
  - leveraging InDesign professional typesetting and formatting capabilities for structured, maybe partially generated, content
  - InDesign included in highly formalised and automatised publication workflows
- Structural content added to documents prepared manually:
  - database records
  - standard text fragments like “Legal Notes” from common source

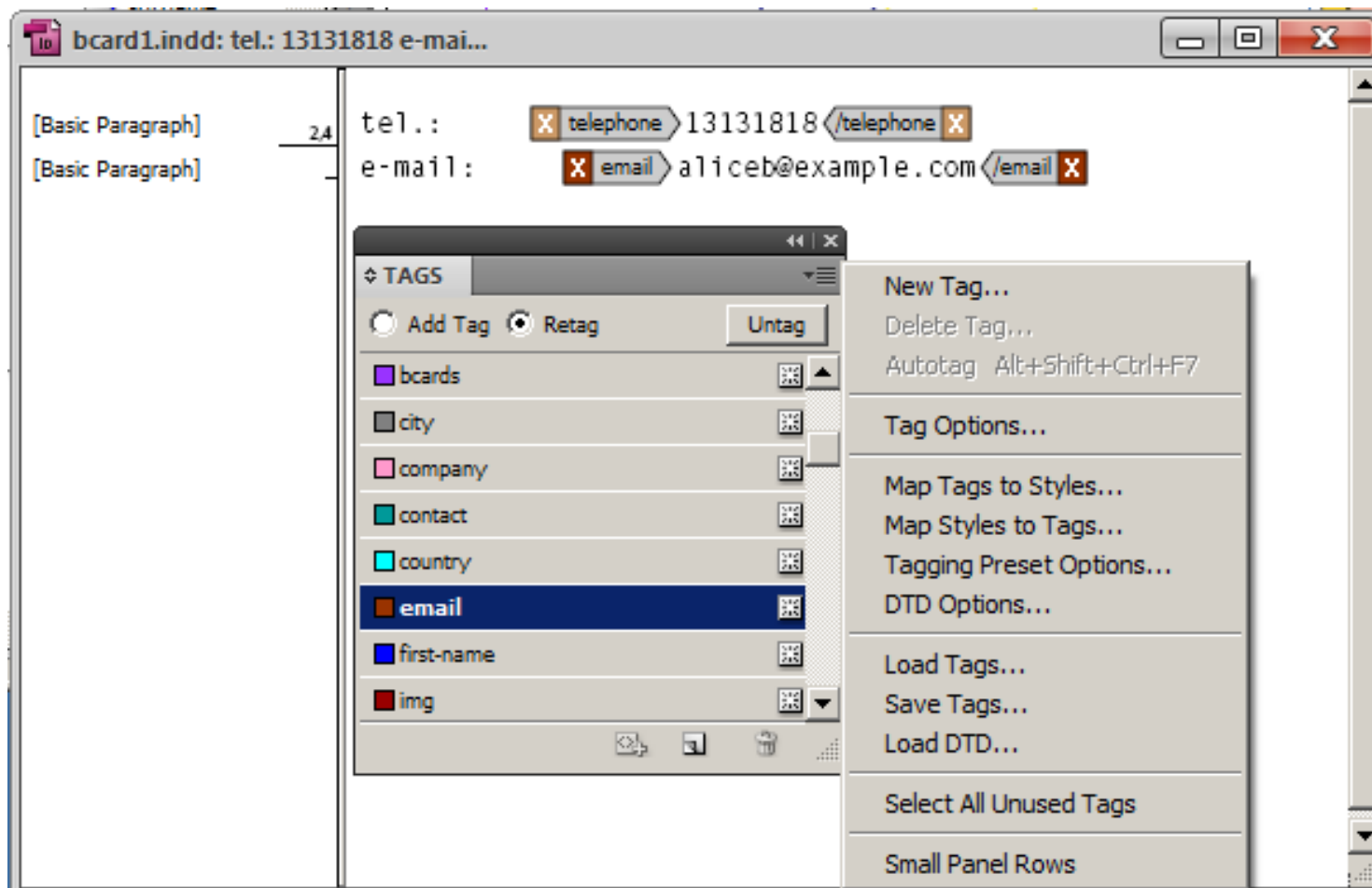
# Structure pane and Layout view for tagged, multi-frame document



# Structure pane context menu



# Story Editor and Tags panel with Tags menu expanded



# XML content in InDesign

- May exist in any InDesign document
- Whole XML content can be found in **Structure** pane  
**View > Structure > Show Structure**
- Parts of XML content may occur in text stories and be visible in text frames
  - “tagged” text frames
  - parts (or even whole) of XML content may be hidden – not included in any text frame of document
- XML tags in text are indicated with:
  - small colour markers in main document view (“layout view”)
  - more verbose markers similar to actual XML tags in **Story Editor**

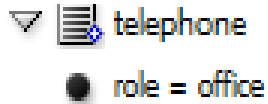
# XML content in InDesign – summary

- Where does it come from?
  - importing external XML documents
  - tagging text content (manually or basing on styles)
- How do we use it?
  - exporting to XML documents
  - distributing parts of XML tree to (many, in general) text frames and stories
- Some restrictions:
  - one XML tree for document (even if many text frames and stories)
  - one XML element cannot be included in more than one place in the document (“1-1 mapping”)



# Attributes of elements

- Attributes available only in **Structure** pane:



- Adding, removing or editing available from **Structure** pane menu or context menu
- Restrictions:
  - attributes not available in text flow
  - attribute values cannot be printed
- Attributes usage in InDesign:
  - imported from / exported to XML
  - can be used during external XML processing
  - **href** attribute for image locations
  - special InDesign attributes (in separate namespace) for style annotation and table metadata

# Comments and processing instructions

- Comments and processing instructions available in **Structure** pane
  - adding, removing, editing
- Visible, but not editable in **Story Editor** and layout view
- Usage of comments and processing instructions:
  - imported from / exported to XML
  - can be used during external XML processing

# Mixing XML and unstructured content

- Unstructured frames
  - unrelated to structured content
  - may be used as header and footer or for similar approaches
  - or in documents merged from many structured and unstructured sources
  - not taken into account on export and import
- Plain text within structured story
  - interleaving with elements
  - on export, placed within parent element for whole story

# Exporting XML

- File > Export > (choose XML as file type)  
or Export XML from Structure pane context menu
- Saves the structured content of document in XML file
- Unstructured text frames omitted

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bcard>
  <person><first-name>Alice</first-name>
    <!--This is a comment--><surname>Blonde</surname>
  </person>
  <address role="office">...</address>
  <contact>
    tel.: <telephone role="office">13131818</telephone>
    e-mail:<email role="office">aliceb@example.com</email>
  </contact>
</bcard>
```

# Using styles and XML together

- Manual formatting and style application
- Mapping tags to styles
- Mapping styles to tags
- Style information in XML attributes

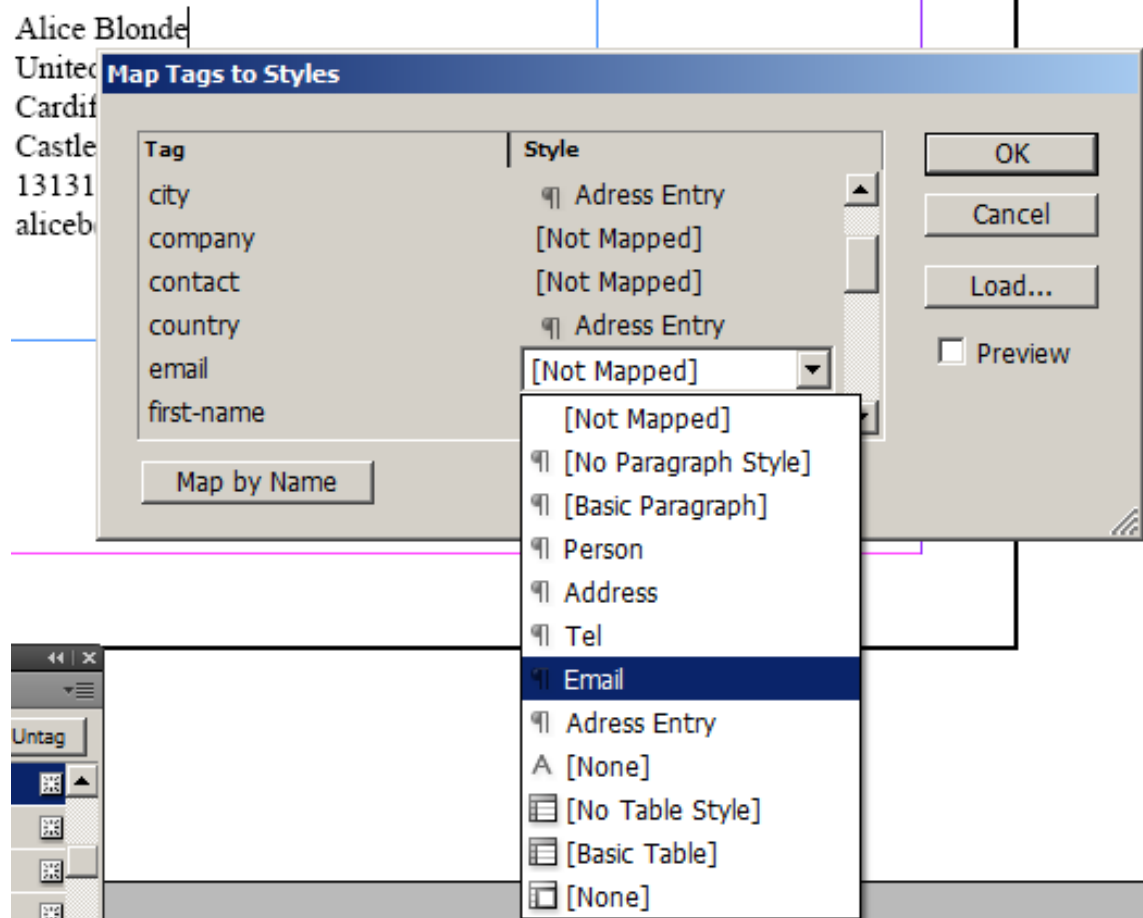
# Manual formatting and style application

- XML content formatted as any text content
  - manual formatting
  - formatting with styles
- Recommended for:
  - one-time project (formatting not intended to be used again)
  - short text
  - non-repeating XML elements or formatting unrelated to XML structure

# Mapping tags to styles

- Automatic application of styles to elements
  - paragraph, character, table, and table cell styles applicable
- Styles reapplied in all tagged stories of document
- Elements with the same name receive the same chosen style
- Styles have to be already defined
- Options:
  - **Preview** – changes visible in layout view before accepting
  - **Map by Name** – uses styles of the same name for elements, where applicable
  - **Load** – loads styles from external file

# Mapping tags to styles





# Mapping tags to styles – example

- Sample document before and after applying styles

Alice Blonde  
United Kingdom  
CardiffCF10  
Castle St13  
13131818  
aliceb@example.com

Alice Blonde

United Kingdom  
CardiffCF10  
Castle St13

- 13131818
- *aliceb@example.com*

# Mapping tags to styles – remarks

- Benefits:
  - fast formatting of large documents
  - consistent formatting
  - easy style enhancements in future
- Difficulties and discomforts:
  - styles have to be defined manually before mapping
  - special characters – paragraph breaks, spaces etc. – have to exist in structured content before formatting
    - one paragraph style used for many XML elements in case that those elements reside in the same source line
  - unneeded indents and line breaks from XML cannot be eliminated easily

# Style information in XML attributes

- Attributes understood by InDesign may contain information which style to apply
    - attributes within Adobe InDesign namespace:  
`aid = http://ns.adobe.com/AdobeInDesign/4.0/`
    - **aid:pstyle** - paragraph style
    - **aid:cstyle** - character style
  - InDesign applies style according to them on import, if provided
  - Benefits:
    - styles applied on import, user interaction not required
    - different styles may be used for the same tag name
- ! XSLT preprocessing may prepare attributes

# Style attributes - example

```
<bcard
  xmlns:aid="http://ns.adobe.com/AdobeInDesign/4.0/">
  <person aid:pstyle="Person">
    <first-name>Alice</first-name>
    <surname aid:cstyle="SName">Blonde</surname>
  </person>
  ...
</bcard>
```

[Alice] *[Blonde]*

[United Kingdom]

[Cardiff][CF10]

[Castle St][13]

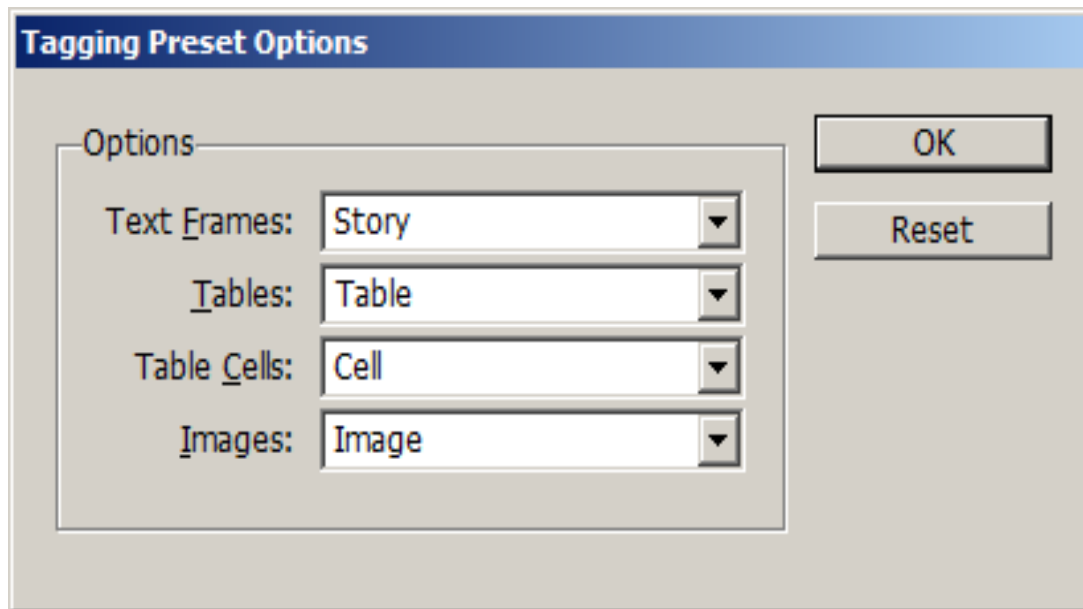
- [13131818]
- [aliceb@example.com]

# Mapping styles to tags

- Each fragment marked with a given style placed within corresponding element
  - Elements have to be known – import DTD or XML before
  - Benefits and applications:
    - fast tagging of unstructured content
  - Difficulties and risks:
    - removes current structure of document;  
to be applied carefully to structured documents
    - requires document to be properly formatted with styles, where styles denote structure (and semantics in the best case)
    - styles structure should correspond to desired XML structure – hard to achieve for elaborated, nested XML structures
- ! XSLT postprocessing may be used for correcting structure

# Automatic tagging

- Tags > Autotag
- Another way for fast tagging whole document
- More coarse-grained: tags applied to frames and objects
- Tags > Tagging Preset Options (default values given below):



# Import XML – two main approaches

- Content-first approach:
  - import content
  - then take care of it:
    - distribute content to text frames
    - format (by e.g. mapping tags to styles)
- Placeholders approach:
  - prepare document with stub content distributed and formatted as desired
  - then import (merge) XML and get it distributed and formatted automatically

# Importing XML – content first approach

- Benefits:
  - real document visible while preparing layout and formatting
  - fast final result
- Disadvantages:
  - manual work to do each time document is imported
- Reasonable usage:
  - one-time process
    - e.g. importing structured content into larger, unstructured document, produced on special demand...
  - preliminary step while preparing template in placeholder approach



# Importing XML – placeholders approach

- Benefits:
  - cheap application to arbitrary many documents
  - repeatable, predictable results
- Disadvantages:
  - more preliminary work required
- Reasonable usage:
  - repeatable tasks
  - part of (semi-)automatised publication process when (part of) data comes from external XML
  - InDesign document as (very advanced...) stylesheet
    - saved as template allows for easy fresh documents creation

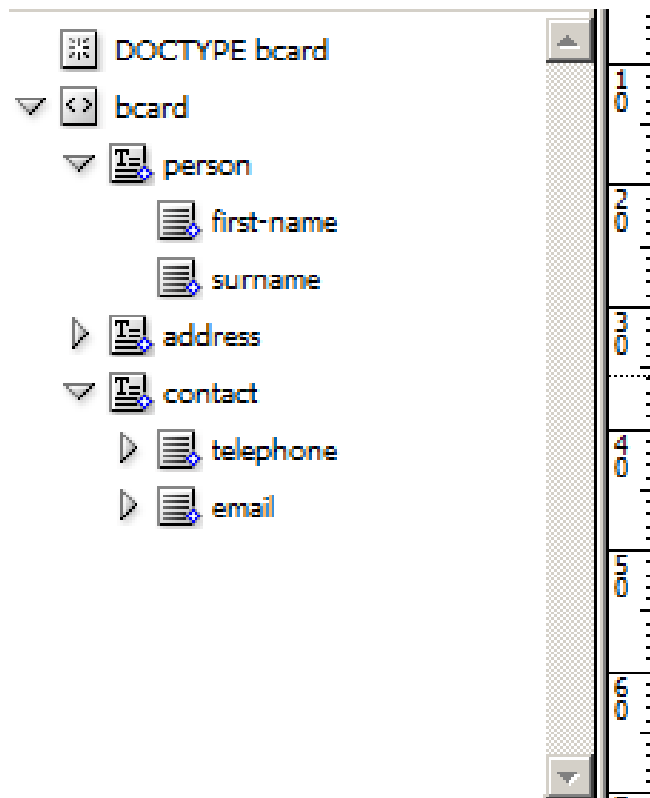
# Preparing placeholders for XML import

- Prepare stub structure and content:
    - one instance of each XML structure we want to handle
    - example content; representative in length of texts etc.
    - real document may be used
      - usually should be simplified – repeated elements removed etc.
  - Distribute content fragments to text frames and format them accordingly
    - use **Map Tags to Styles** as much as possible
    - add spaces, line breaks, tabs
    - add text labels and other static content
- ! Add static text and characters outside text XML elements to avoid removing them on import

# Preserving labels and special characters

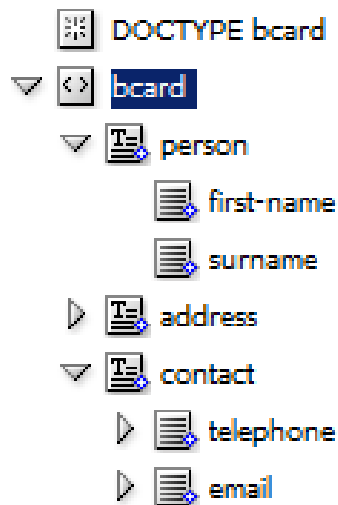
- Good XML contains only actual data
  - no labels, separators and other redundant content
- Adding labels, spaces, etc. – part of formatting process
  - need to put static content among placeholders, so that importing does not erase them
- Put static content in places where XML to be imported does not contain any (non-whitespace) content
  - usually before, after, or between text elements
  - sometimes inside empty elements
- Select import option
  - Do not import contents of whitespace-only elements

# Placeholders prepared – example



[First][Last]	
	Country [code][City] Street[123]
tel.: [123456789] e-mail: [xxxx@bbbbbb.com]	

# XML imported into placeholders



[Alice][Blonde]	
	[United Kingdom] [code][Cardiff] [CF10][Castle St][13]
tel.: [13131818] e-mail: [aliceb@example.com]	

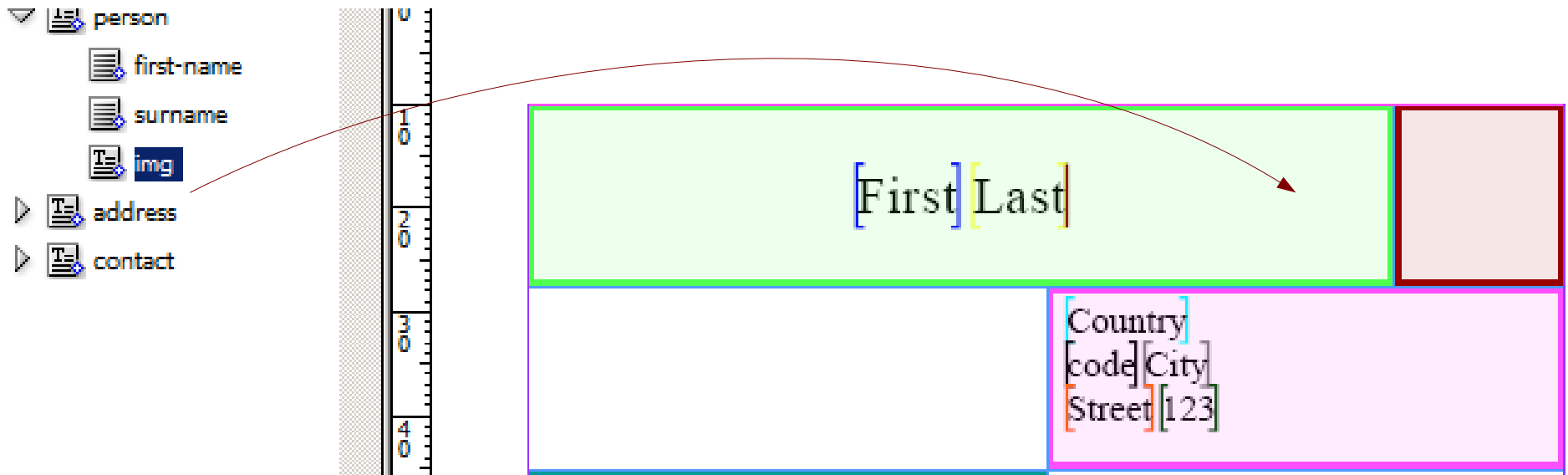
# Handling repeating elements

- Repeated structures typical for XML content
  - especially in case of database records saved in XML format
- Prepare one placeholder for repeating element (the “record”)
- Set **Clone repeating text elements** option on import
- Placeholders get cloned
  - with actual content inserted, naturally

# Tagging images

- Images may be tagged as XML elements
- Works best if element:
  - is empty
  - allows for **href** attribute
- Tagging:
  - **Tag** from image frame context menu
  - or drag element onto image frame
- **href** attribute:
  - inserted automatically by InDesign when adding or tagging image
  - exported in XML
  - used to load image on XML import, if image placeholder used for corresponding element in document stub / template
  - file: URLs, e.g. `file:///path/image.png`

# Preparing placeholder for image





# Image imported into placeholder

DOCTYPE bcard

bcard

person

first-name

surname


img

href = file:///C:/Users/Patryl

address

contact

[Alice][Blonde]



United Kingdom

CF10

Cardiff

Castle St

13

tel.: 13131818

e-mail: aliceb@example.com

# Tables in structured content

## Goal

- Importing and exporting tables between InDesign and XML

## Available solutions

- Tables tagged with custom tags
  - 2 levels of structure: table and cell
    - no rows, groups, columns, ...
  - number of columns known to InDesign...
    - placeholder with given number of columns
    - special InDesign attributes
  - mapping table styles to / from tags
- CALS tables
  - more structure levels: table, tgroup, tbody, row, entry
  - attributes with metadata

# Exporting CALS tables

- Preconditions:
  - table within tagged text frame
  - table not tagged
- Export XML  
with option **Export Untagged Tables as CALS XML**
- Such table can be then imported with option **Import CALS Tables as InDesignTables**

Text above table.		
Person	Position	Salary
Alice Blone	manager	2500
John Brown	secret agent	2300
Rebeca Green	secretary	5200
Total		10000
Text below table.		

# Example table exported as CALS XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Root><Story>Text above table.
<table frame="all"> <tgroup cols="3">
<colspec colname="c1" colwidth="119.40551181102364pt"></colspec>
...
<thead><row>
  <entry align="left"   valign="top">Person</entry>
  <entry align="left"   valign="top">Position</entry>
  <entry align="left"   valign="top">Salary</entry>
</row></thead>
<tfoot><row>
  <entry align="left"   valign="top">Total</entry>
  <entry align="left"   valign="top"></entry>
  <entry align="right"  valign="top">10000</entry>
</row></tfoot>
<tbody><row>
  <entry align="left"   valign="top">Alice Blone</entry>
  <entry align="left"   valign="top">manager</entry>
  <entry align="right"  valign="top">2500</entry>
</row>
...</tbody>
</tgroup>
</table>Text below table.</Story></Root>
```

# Importing custom tags as tables

- Solution for importing externally-provided XML (e.g. records from database) and placing data in tables in InDesign documents
- Prepare tagged table as placeholder in template document
  - one tag for whole table
  - tags for individual cells inside table
  - number of columns should match (XML and template)
- Import with option  
Import text elements into tables if tags match

# Custom XML content to be imported as table

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Root><Story>Text above table.
<salaries>
  <person>Person</person>
  <position>Position</position>
  <salary>Salary</salary>

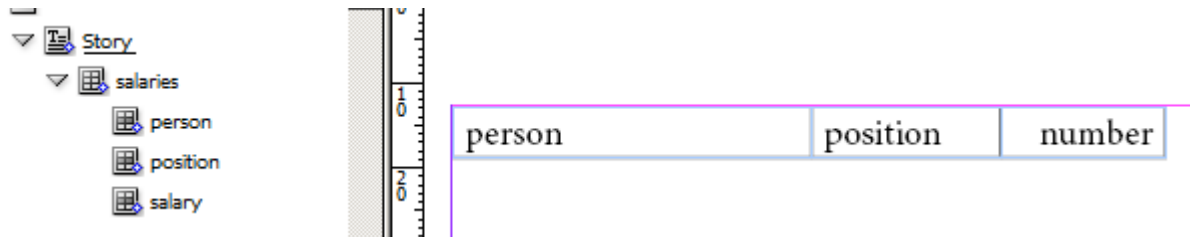
  <person>Alice Blone</person>
  <position>manager</position>
  <salary>2500</salary>

  <person>John Brown</person>
  <position>secret agent</position>
  <salary>2300</salary>

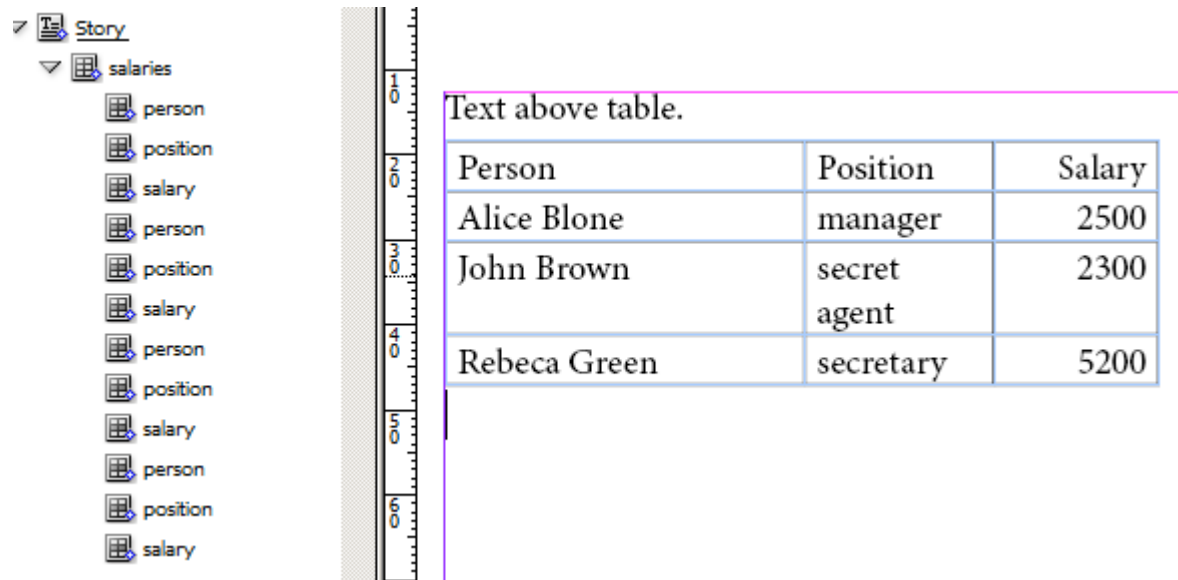
  <person>Rebeca Green</person>
  <position>secretary</position>
  <salary>5200</salary>
</salaries>
</Story></Root>
```

# Placeholder for table

## Placeholder for table before and after import



person	position	number
--------	----------	--------



Text above table.

Person	Position	Salary
Alice Blone	manager	2500
John Brown	secret agent	2300
Rebeca Green	secretary	5200

# InDesign attributes in table tags

- Flexible table tagging
  - custom element names
  - table-related metadata in InDesign attributes
- No table placeholder needed in template document
  - table placed within tagged text frame automatically interpreted as table on import
- Additional features:
  - number of columns set individually for each table
  - table header and footer
  - size and style of table and cells



# Example table with InDesign attributes

```
<salaries xmlns:aid="http://ns.adobe.com/AdobeInDesign/4.0/"
  aid:table="table" aid:trows="5" aid:tcols="3">
  <Cell aid:table="cell" aid:thead="" aid:crows="1" aid:ccols="1"
    aid:ccolwidth="119.4">Person</Cell>
  <Cell aid:table="cell" aid:thead="" aid:crows="1" aid:ccols="1"
    aid:ccolwidth="70.8">Position</Cell>
  <Cell aid:table="cell" aid:thead="" aid:crows="1" aid:ccols="1"
    aid:ccolwidth="63.5">Salary</Cell>

  <Cell aid:table="cell" aid:crows="1" aid:ccols="1"
    aid:ccolwidth="119.4">Alice Blone</Cell>
  <Cell aid:table="cell" aid:crows="1" aid:ccols="1"
    aid:ccolwidth="70.8">manager</Cell>
  <Cell aid:table="cell" aid:crows="1" aid:ccols="1"
    aid:ccolwidth="63.5">2500</Cell>
  ...

  <Cell aid:table="cell" aid:tfooter="" aid:crows="1" aid:ccols="1"
    aid:ccolwidth="119.4">Total</Cell>
  <Cell...></Cell>
  <Cell ...>10000</Cell>
</salaries>
```

# IDML file format

- IDML (for InDesign) and ICML (for InCopy)
  - interrelated file formats for storing and interchanging documents
  - open, XML-based
  - replace INX and INCX formats used up to CS 3
- IDML document – ZIP archive of files
  - XML files for structure, metadata and text content
  - binary files for embedded fonts, images, etc.
- Directories within archive:



```
graph LR; Root[ ] --- MasterSpreads; Root --- META-INF; Root --- Resources; Root --- Spreads; Root --- Stories; Root --- XML;
```

A diagram showing the directory structure of an IDML archive. It consists of a vertical line on the left with horizontal branches to the right, each labeled with a directory name: MasterSpreads, META-INF, Resources, Spreads, Stories, and XML.

# Handling IDML files – potential benefits

- Theoretical ability of handling IDML files without InDesign
  - custom applications, scripts, XSLT
- Potential benefits:
  - reading contents created with InDesign without InDesign
  - automatised production of final documents ready to be opened by InDesign (or InDesign Server) and printed / published
- Challenges:
  - complex, internally-related format
  - preparation of scripts producing proper documents would most probably require:
    - heavy InDesign usage
    - “reverse engineering”
    - trial-and-error cycles repeated many times

# IDML content overview – root

- `designmap.xml` – overall structure of document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<?aid type="document" featureSet="257" product="7.5(142)" ... ?>
<Document Self="d" ActiveLayer="ub3" CMYKProfile="$ID/" RGBProfile="$ID/"
  DOMVersion="7.5" StoryList="uc4 ue4 u32c u410 u42a u9c" ZeroPoint="0 0"
  xmlns:idPkg="http://ns.adobe.com/AdobeInDesign/idml/1.0/packaging" ...>
  <Language Self="Language/$ID/English%3a UK" Name="$ID/English: UK"
    SingleQuotes="'" DoubleQuotes=""" PrimaryLanguageName="$ID/English"/>
  <idPkg:Graphic src="Resources/Graphic.xml"/>
  <idPkg:Fonts src="Resources/Fonts.xml"/>
  <idPkg:Styles src="Resources/Styles.xml"/>
  ...
  <TextVariable Name="Creation Date" VariableType="CreationDateType"...>
    <DateVariablePreference TextBefore="" Format="dd/MM/yy"
TextAfter=""/>
  </TextVariable>
  <idPkg:MasterSpread src="MasterSpreads/MasterSpread_ubd.xml"/>
  <idPkg:Spread src="Spreads/Spread_ub6.xml"/>
  <idPkg:BackingStory src="XML/BackingStory.xml"/>
  <idPkg:Story src="Stories/Story_u42a.xml"/>
  ...
```

# IDML content overview – tags

- `XML/Tags.xml` – list of tags available for document

```
<idPkg:Tags DOMVersion="7.5"
xmlns:idPkg="http://ns.adobe.com/AdobeInDesign/idml/1.0/packaging" >
  <XMLTag Self="XMLTag/address" Name="address">
    <Properties>
      <TagColor type="enumeration">Yellow</TagColor>
    </Properties>
  </XMLTag>
  <XMLTag Self="XMLTag/city" Name="city">
    <Properties>
      <TagColor type="enumeration">Cyan</TagColor>
    </Properties>
  </XMLTag>
  ...

```

# IDML content overview – backing story

- `XML/BackingStory.xml` – structure tree of document
  - contains content not placed in text frames
  - points to stories placed in text frames

```
...
<XMLElement Self="di2" MarkupTag="XMLTag/common-info">
  <XMLElement Self="di2i3i21i23" MarkupTag="XMLTag/email">
    <Content>aliceb@example.com</Content>
    <XMLAttribute Self="di2i3i21i23XMLAttributenrole"
      Name="role" Value="office"/>
  </XMLElement>
...
  <XMLElement Self="di2i10" MarkupTag="XMLTag/license"
XMLContent="ue4"/>

  <Link Self="u327" LinkResourceURI="file:C:/.../common_info.xml" ... />
</XMLElement>
...
```

# IDML content overview – stories

- XML/Stories/Story\_u???.xml – content of text stories
  - tagged and untagged content
  - XML tags, style information, and all other metadata stored

```
<Story Self="ue4" AppliedTOCStyle="n" TrackChanges="false"
  StoryTitle="$ID/" AppliedNamedGrid="n">
  <XMLElement Self="di2i10" MarkupTag="XMLTag/license" XMLContent="ue4">
  ...
  <ParagraphStyleRange AppliedParagraphStyle="ParagraphStyle/body">
    <CharacterStyleRange AppliedCharacterStyle="CharacterStyle/$ID/
      [No character style]">
      <Content>First line</Content><Br/><Content>The first successful
        locomotives were built by Cornish inventor </Content>
    </CharacterStyleRange>
    <CharacterStyleRange AppliedCharacterStyle="CharacterStyle/person">
      <Content>Richard Trevithick</Content>
    </CharacterStyleRange>
  ...
```