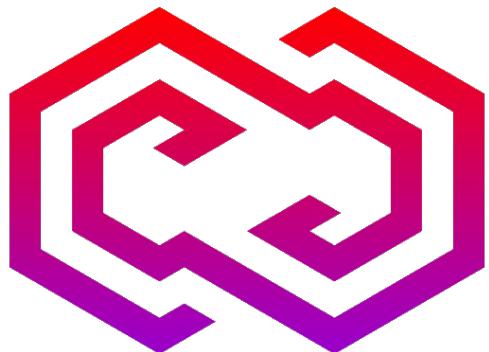




Markup UK

2021 Proceedings

A Conference about XML and Other Markup Technologies



FUSIONDB

le^rtex
publishing services



Markup UK

A Conference about XML and Other
Markup Technologies
<https://markupuk.org/>

Markup UK Conferences Limited
is a limited company registered in England
and Wales.
Company registration number: 11623628
Registered address: 24 Trimworth Road,
Folkestone, CT19 4EL, UK
VAT Registration Number: 316 5241 25

Organisation Committee

Geert Bormans
Tomos Hillman
Ari Nordström
Andrew Sales
Rebecca Shoob

Programme Committee

Syd Bauman – Northeastern University
Digital Scholarship Group
Achim Berndzen – <xml-project />
Abel Braaksma – Abrasoft
Peter Flynn – University College Cork
Tony Graham – Antenna House
Michael Kay – Saxonica
Jirka Kosek – University of Economics,
Prague
Deborah A. Lapeyre – Mulberry
Technologies
David Maus – State and University Library
Hamburg
Adam Retter – Evolved Binary
B. Tommie Usdin – Mulberry Technologies
Norman Walsh – MarkLogic
Lauren Wood – XML.com

Thank You

Evolved Binary
le-tex Publishing Services
Saxonica
OxygenXML
Exeter
Adam Retter
Tony Graham
...and our long-suffering partners

Sister Conferences

Balisage
The Markup Conference

 xmlprague


summer school

 Declarative
Amsterdam

Markup UK 2021 Proceedings

by B. Tommie Usdin, David Maus,
Alain Couthures, Michael Kay, Erik
Siegel, Debbie Lapeyre, Karin Bredenberg,
Jaime Kaminski, Robin La Fontaine,
Nigel Whitaker, Steven Pemberton, Tony
Graham and Liam Quin

The organisers of Markup UK would like to
thank Antenna House for their expert and
unstinting help in preparing and formatting
the conference proceedings, and their
generosity in providing licences to do so.

Antenna House Formatter is based on the
W3C Recommendations for XSL-FO and
CSS and has long been recognized as
the most powerful and proven standards
based formatting software available. It is
used worldwide in demanding applications
where the need is to format HTML and
XML into PDF and print. Today, Antenna
House Formatter is used to produce
millions of pages daily of technical,
financial, user, and a wide variety of other
documentation for thousands of customers
in over 45 countries.



Markup UK

Encouraging Tag Set Branching without Creating a Briar Patch 6

B. Tommie Usdin.

What's in a Schematron? 18

David Maus.

XSLTForms for the '20s 26

Alain Couthures.

<transpile from="Java" to="C#" via="XML" with="XSLT"/> 34

Michael Kay.

Comprehensible XML 50

Erik Siegel.

How Much Tag Set Documentation is Needed? 62

Debbie Lapeyre.

2021 The Future of Distributed Markup Systems or ‘Help my package has become too large!’ 100

Karin Bredenberg, Jaime Kaminski.

An improved diff3 format using XML: diff3x 108

Robin La Fontaine, Nigel Whitaker.

On the Design of a Self-Referential Tutorial 124

Steven Pemberton.

“FYI we’re not looking to go to print” 134

Tony Graham.

CSS From XSLT 146

Liam Quin.



Encouraging Tag Set Branching without Creating a Briar Patch

B. Tommie Usdin, Mulberry Technologies, Inc.

©2021 Mulberry Technologies, Inc.

Customizing a tag set can be an easy way to get the vocabulary you need. It can also be a journey filled with dead ends, trap doors, and slowly-revealed and difficult to identify problems. Like many public tag sets, JATS (the Journal Article Tag Suite) was designed to be customized. Our original expectation was that individual users would customize it, and while a few have done that to good effect, we have found that the major customizations have been by groups of users. BITS (the Book Interchange Tag Suite), NISO STS (Standards Tag Suite), and Manuscript Exchange Common Approach (MECA) are widely adopted customizations of JATS.

When users customize a tag set they expect to be able to use the existing infrastructure associated with that tag set, making changes to accommodate the changes they made. They often expect to intermingle their new documents with documents tagged to the original tag set and perhaps with documents tagged to other customizations of the source tag set. They expect this to work gracefully, easily, seamlessly. Sometimes it does, but sometimes it does not!

The “JATS Compatibility Meta-Model Description” was developed to help people who customize JATS create tag sets to create models that will coexist peacefully with existing JATS documents and with documents tagged to other JATS customizations.

It seems unlikely that the particulars of the JATS Compatibility Model will apply to other tag sets, but the principles behind the Meta-Model might be useful to other groups thinking about ways to make their families of tag sets flexible and compatible.

1. Customizing Tag Sets

- ◊ Not new
- ◊ A good idea (usually)
- ◊ Saves work, time, money
- ◊ Promotes document interchange
- ◊ Can go horribly wrong

2. A Little History (over simplified)

2.1. Generic Markup

- ◊ Tag what it is not what it should look like
- ◊ Province of system (e.g., IBM Starter Set was tag set for IBM Script)
- ◊ Didn't meet all needs
- ◊ Attempt to make "universal" list of tags
- ◊ Impossible
- ◊ Developed "name your own tag" systems

2.2. Bespoke Vocabularies

- ◊ Markup projects started with "Task 1 - Create the Tag Set"
- ◊ Expensive and time consuming
- ◊ Some considered trade secrets
- ◊ Required custom infrastructure
 - ◆ Formatter customization cost as much as (or more than) vocabulary
 - ◆ Interchange required custom transform for each destination
- ◊ Constant expert maintenance

2.3. Public Vocabularies

- ◊ Developed to
 - ◆ Promote interchange
 - ◆ Lower barrier to entry
- ◊ Enabled
 - ◆ Shared tools
 - ◆ Lower cost services
 - ◆ Shared expertise, software

2.4. Need for Customization Soon Obvious

- ◊ Vocabulary maintainers flooded with requests
- ◊ Tag sets began to grow
- ◊ Tag abuse grew
- ◊ Local customizations were clever but often hacks

3. Vocabularies Build In Customization Tools

3.1. “Symposium on Markup Vocabulary Customization”

Proceedings at: <https://www.balisage.net/Proceedings/vol24/cover.html>

- ◊ Akoma Ntoso
- ◊ DITA
- ◊ DocBook
- ◊ JATS
- ◊ TEI

3.2. Debates about Value of Customization

A few examples from JATS:

- ◊ “Superset Me—Not: Why the Journal Publishing Tag Set Is Sufficient if You Use Appropriate Layer Validation”, Alexander B. Schwarzman. (2010) <https://www.ncbi.nlm.nih.gov/books/n/jatscon10/schwarzman/>
- ◊ “Why Create a Subset of a Public Tag Set”, Deborah Aleyne Lapeyre. (2010) <https://www.ncbi.nlm.nih.gov/books/n/jatscon10/lapeyre/>
- ◊ “When the ‘One Size Fits Most’ tagset doesn’t fit you”, Tommie Usdin. (2013) <https://www.ncbi.nlm.nih.gov/books/n/jatscon13/usdin/>
- ◊ “JATS Subset and Schematron: Achieving the Right Balance”, Alexander B. Schwarzman. (2017) <https://www.ncbi.nlm.nih.gov/books/n/jatscon17/schwarzman/>

3.3. Customization Mechanisms Vary

- ◊ DITA protects content from uncustomized tools
- ◊ TEI provides a web-based customization tool (Roma)
- ◊ Akoma Ntosa assumes users familiar with HTML
- ◊ JATS assumes customization by XMLers

3.4. Customizations

- ◊ Useful
- ◊ Convenient
- ◊ Appropriate
- ◊ Can go wrong

4. Experience with JATS Customizations

- ◊ JATS designed to be customized from beginning
- ◊ Provided “Modifying This Tag Set” in documentation: <https://jats.nlm.nih.gov/archiving/tag-library/1.2/chapter/implementor.html>

- ◊ Expected individual users to customize

4.1. Many Customizations by Groups of Users

Standards committees wrote:

- ◊ BITS (the Book Interchange Tag Suite)
- ◊ NISO STS (Standards Tag Suite)
- ◊ Manuscript Exchange Common Approach (MECA)

4.2. User Expectations for a Customization

- ◊ Existing infrastructure will work
- ◊ Only change parts directly related to new content
- ◊ Only modify to accommodate new requirements
- ◊ Intermix old and new documents
- ◊ Easy, graceful, seamless integration

4.3. User Experience

- ◊ Sometimes it “just works” as expected
- ◊ Sometimes problems appear on initial testing
- ◊ Sometimes problems pop up later
- ◊ Some problems easy to identify & fix
- ◊ Some problems very difficult to identify
- ◊ Some problems found too late to fix

5. JATS Compatibility Model

“Building JATS-Compatible Vocabularies Draft 0.10” — formerly called “JATS Compatibility Meta-Model”

- ◊ Developed to help people who customize JATS create tag sets to create models that will coexist peacefully with existing JATS documents and with documents tagged to other JATS customizations
- ◊ Particulars unlikely to apply to other tag sets
- ◊ Principles probably do apply to others

Current version at: <https://www.niso.org/publications/jats-compatibility-model>

5.1. JATS Compatibility Model in 2 Parts

- ◊ Design Principles
- ◆ about models as a whole

- ◆ philosophical more than technical
- ◊ Compatibility Properties
 - ◆ about individual structures
 - ◆ some highly technical
 - ◆ some automatically checkable

5.2. Design Principles

5.2.1. Respect the Semantics

- ◊ The *most important* rule
- ◊ Use named structures to mean the same thing as source vocabulary
- ◊ Do not “reuse” structures

This applies to *ALL* vocabularies!

5.2.2. Prevent Semantic Mismatch

Homonyms are *destructive* in tag sets

For example, in a collection of city regulations:

- ◊ you may not need the JATS <license> element (Set of conditions under which the content may be used, accessed, and distributed.)
- ◊ you may need to know if a dog, bicycle, or bar needs a license to operate
- ◊ Create a NEW structure, perhaps <city-license> or <permit-license> or <MyCity:license>
- ◊ DO NOT repurpose <license>

5.2.3. Linking Direction

- ◊ Links may go in either direction, or both
- ◊ Match the link philosophy of the source vocabulary
- ◊ In JATS:
 - ◆ links go from many to one
 - ◆ citations point to references, because a reference may be cited multiple times
 - ◆ cross-references point to the table, figure, box, etc.
- ◊ Hint: link *display* can go in either or both directions from one-way ID/IDREF in XML source

5.2.4. Section Model

- ◊ Headed sections are the defining structure of textual documents
- ◊ Section tagging is fundamental to prose markup

5.2.5. Common Section Models

- ◊ Mix of paragraphs and styles/levels of headings (H1, H2, ...) — (HTML, most word processors)
- ◊ Explicit section levels, with level-designated headings — (sec1 contains head1, paras; sec2 contains head2, paras; sec3 contains head3, paras)
- ◊ Nested explicit section levels, with level-designated headings — (sec1 contains head1, paras, & sec2; sec2 contains head2, paras, & sec3 ...)
- ◊ Recursive sections with undesignated headings — (sec contains head, paras, & sec; sec contains head, paras, & sec ...)

(IMO: the only wrong/bad choice on section model is to allow more than one of these in a document set.)

Use the section model of your source vocabulary — JATS and JATS-compatible models use recursive sections

5.2.6. Subsetting

How much and what may be removed is not obvious

- ◊ Applications may depend on presence of some content
 - ◆ Removing structures may break applications
 - ◆ Electronic display may rely on section titles; making them optional may kill navigation
 - ◆ Display of figures may rely on reference to them
- ◊ Subsetting is Desirable
 - ◆ Requiring unavailable content leads to tag abuse
 - ◆ Presence of unused structures
 - ◆ increases learning curve, cost for tools and customizations
 - ◆ reduces tagging consistency and interchange

5.2.7. JATS: Subsetting Always Allowed

- ◊ Most controversial of guidelines
- ◊ We are already there: in JATS Archiving everything is optional except one ID
- ◊ In JATS compatible models it is OK to:
 - ◆ Remove elements from an “or” group or sequence
 - ◆ Make required elements optional (or remove entirely)
 - ◆ Remove values from an attribute value list
 - ◆ Remove attribute from element

5.3. Compatibility Properties in JATS Guidelines

- ◊ Each element/attribute must match on all (relevant) properties
- ◊ Note: JATS developed before these guidelines; JATS itself does not follow all recommendations

5.3.1. Don't Munge Elements and Attributes

- ◊ Some info could be element or attribute
- ◊ Some XMLers passionate about element/attribute choice
- ◊ XSLT makes converting elements ↔ attributes easy
- ◊ JATS-Compatible tag set should:
 - ◆ follow JATS use for element or attribute content, or
 - ◆ create a new name for new element or attribute
- ◊ JATS has <phone> element. If want phone number as an attribute value, call it something OTHER than @phone
- ◊ JATS has @rationale attribute. If want an element for rationale, call it something OTHER than <rationale>

5.3.2. Section-like

- ◊ JATS sections have specified structure
 - ◆ Section-Head (labels, titles, licensing, section-metadata)
 - ◆ Section-Blocks (paragraphs, tables, figures, etc.)
 - ◆ Sections (sections may contain sections, AFTER section-blocks)
 - ◆ Section-Tail (typically notes, footnotes, ref-list)
- ◊ Some section-like structures have full section model — (section, abstract, boxed-text, appendix)
- ◊ Some section-like structures have partial section model — (statement)
- ◊ Compatible models for these structures must have section structure or subset of it

5.3.3. Tagset-Specific Constructs

- ◊ Many tagset have characteristic/special constructs
- ◊ Use them as defined or don't use them at all
- ◊ DITA has maps, keys, and conref
- ◊ JATS has alternatives

5.3.4. Alternatives

- ◊ Used in JATS to wrap several equivalent structures

- ◊ Typically, only one is counted or displayed
- ◊ One may be preferred
- ◊ May be alternatives for different media, languages, resolutions, formats

5.3.5. Display Alternatives Example

- ◊ A display formula might contain an alternatives that contains:
 - ◆ a graphic
 - ◆ a MathML expression
 - ◆ the textual form of the expression
- ◊ 3 ways to express the same information
- ◊ MathML may be used for print
- ◊ Screen display may use graphic
- ◊ Textual form used as alt-text for non-visual users

5.3.6. Whitespace Handling in XML Documents

- ◊ Whitespace is spaces, line breaks, carriage returns, tabs
- ◊ How XML handles whitespace seems obscure
- ◊ Very important
 - ◆ Sometimes whitespace is “significant”, sometimes not
 - ◆ Many XML processors behave differently when whitespace is/isn’t significant

5.3.7. Maintain Whitespace Type of Source Vocabulary

(a short dive into XML esoterica)

- ◊ 3 whitespace types in XML
 - ◆ Not significant (element-only contexts)
 - ◆ Collapse to 1 space (most text, including mixed content)
 - ◆ Preserve (pass exactly; only when so specified)
- ◊ **Compatible tag set must not change whitespace type for any element**
- ◊ Can be source of much mysterious weirdnesses — and very hard to find.

5.3.8. ID IDREF

Identifier pattern should follow source vocabulary

- ◊ Attributes that are typed as ID must remain ID
- ◊ Attributes that are typed as IDREF must remain IDREF (or IDREFS)

Remember, attributes that contain IDs or IDREFs can have any name — JATS
 @continued-from is an IDREF attribute

5.3.9. Property Catalog in JATS Guidelines

- ◊ Checklist
- ◊ Catalog lists elements, attributes, all properties
- ◊ Shows several JATS-related tag sets

JATS Structure Name			ID or IDREF	Whitespaces	Section-like*	Alternatives*	JATS Archiving 1.2	JATS Publishing 1.2	JATS Authoring 1.2	BITS 2.0	NISO STS 1.0
cellpadding	attribute						X	X	X	X	X
cellspacing	attribute						X	X	X	X	X
chapter-title	element			D			X	X	X	X	X
char	attribute						X	X	X	X	X
charoff	attribute						X	X	X	X	X
chem-struct	element			D			X	X	X	X	X
chem-struct-wrap	element			X	E		X	X	X	X	X
citation-alternatives	element	X		E			X	X	X	X	X
city	element			D			X	X	X	X	X
code	element			P			X	X	X	X	X

5.4. Properties not Relevant to Compatibility (as defined)

- ◊ Some properties considered and declared irrelevant for compatibility — [“Circling in on the JATS Compatibility Meta-Model”, Laura Randall, B. Tommie Usdin, Deborah A. Lapeyre, and Jeffrey Beck. <https://www.ncbi.nlm.nih.gov/books/n/jatscon17/randall/>]

5.4.1. Social Behaviour

- ◊ Role in document (e.g., block, inline, metadata)
- ◊ Content (e.g., text, mixed content, toggle/flags)

5.4.2. Metadata vs Data

- ◊ Distinction is problematic
- ◊ Metadata now, data later, or — metadata to me, data to you
- ◊ Book title
 - ◆ title of *this* book is metadata
 - ◆ title of another book, e.g., in reference, is data

5.4.3. Framework or superstructure

- ◊ Big parts of a document

- ◆ root or top level wrapper
- ◆ large divisions of document, e.g., front, body, back
- ◊ Framework in one context becomes body structure in another
 - ◆ article is top level structure in JATS
 - ◆ BITS is designed so that articles can become book chapters with few changes
 - ◆ Standards can be adopted. The adoption document wraps entire standard, with content before and after

5.4.4. Recursion

- ◊ In JATS:
 - ◆ sections may contain sections
 - ◆ paragraphs may not contain paragraphs
- ◊ Disallowing recursion (not allowing sections to contain sections) does no harm
- ◊ Enabling recursion may require changes to display styles but is not destructive

5.5. These Guidelines are *not* Universal

5.5.1. JATS users can Ignore Guidelines

JATS is convenient starting place, but that's all, — Ignore guidelines **IF AND ONLY IF:**

- ◊ Documents will never interact with other JATS documents
- ◊ Users and suppliers do not know JATS
- ◊ Won't adopt JATS tools

5.5.2. Other Groups

- ◊ May want similar guidelines
- ◊ Some of these inapplicable
- ◊ Some we discarded may apply

5.5.3. Ignore and Find Yourself in a Briar Patch

- ◊ If documents co-exist with documents from base tag set
- ◊ If users/vendors know base tag set
- ◊ And you ignore guidelines
- ◊ Expect slow-motion, difficult to diagnose, CHAOS
 - ◆ Surprising search results
 - ◆ Incomprehensible error messages
 - ◆ Formatting errors, especially vanishing or extra spaces and line breaks

6. Note: the Briar Patch in the Talk Title

- ◊ A briar patch is a nasty place
 - ◆ thorn-covered branches
 - ◆ hostile creatures (snakes, porcupine, skunk)
- ◊ Many associate “briar patch” with Uncle Remus and stories of Brer Rabbit and Brer Fox (think: racist Aesop’ fables)
- ◊ Plan was to associate:
 - ◆ incompatabilities with briar patch
 - ◆ compatability with safe path to swimming beach

Some metaphors deserve to die



7. Questions? Comments? Suggestions?

- ◊ Now, in the Q&A window
- ◊ On comments page for the “JATS Compatibility Model” see: <https://www.niso.org/publications/jats-compatibility-model>
- ◊ on JATS-List <http://www.mulberrytech.com/JATS/JATS-List/>
- ◊ or send email to me at <btusdin@mulberrytech.com>



What's in a Schematron?

David Maus, State and University Library Hamburg

This paper presents a preliminary classification of Schematron concepts and discusses the three most important topics covered by ISO Schematron: Rule based validation, schema composition, and reporting. It concludes with rough sketch of a new rule based validation language in the tradition of Schematron that takes the advantages in markup language technology into consideration.

1. Introduction

Schematron is a rule based validation language for structured documents. It was designed by Rick Jelliffe in the late 1990s [JELLIFFE1999] and standardized as part of the ISO Document Schema Language Definitions (DSDL) family in 2006. The last non-ISO version was Schematron 1.6, published in 2002. [JELLIFFE2002] Since the standardization as ISO/IEC 19757-3 two editions have been published. The 2nd edition in 2016, the 3rd edition in 2020. The 3rd edition of ISO Schematron will most likely be the last: The working group was disbanded and, more importantly, ISO removed Schematron from the list of publicly available standards, effectively putting an end to ISO Schematron as an Open Standard.

While losing an Open Standard is annoying it gives the opportunity to look at the specification as an object of an analytic inquiry. Rather than comparing Schematron with other schema languages we could look at ISO Schematron's concepts and ask what function they serve or which topic they cover. This analytic perspective may help structuring a discussion of potential shortcomings or missed opportunities.

2. A preliminary classification of Schematron concepts

Table 1 [18] shows a first classification attempt and identifies six major topics in descending order of importance: Rule based validation, schema composition, reporting, documentation, pattern templating and instance document selection. In the following sections I will provide a brief summary of the topic and the associated concepts, and discuss shortcomings of the current specification.

Table 1. Classification of Schematron concepts

Topic	Schematron concepts
Rule based validation	Assertion, Rule, Pattern, Schema, Query language
Schema composition	Abstract rule, Phase, Inclusion
Reporting	Natural language statement, Properties, Templating language
Documentation	Title, Paragraph, Reference to external documentation

Topic	Schematron concepts
Pattern templating	Abstract pattern
Instance document selection	Subordinate documents

2.1. Rule based validation

2.1.1. Summary

Rule based validation is at the core of Schematron. For Schematron, rule-based validation means that a document is considered to be valid if it passes a specified set of assertion tests.

An *assertion test* is a boolean function over a set of nodes. It is associated with zero or more natural-language statements. Schematron uses two elements to represent assertions: An assertion expressed as a sch:assert element succeeds if the assertion test evaluates to boolean true, an assertion expressed as sch:report element succeeds if the assertion test evaluates to false. Both use a test attribute for the assertion test.

A *rule* is an unordered set of assertions. The assertion elements are grouped into a sch:rule element with a context attribute containing a *context expression*. The context expression is an expression that selects nodes from the instance document. A rule is said to "fire" if its context expression selects a non-empty set of nodes. All assertions of the rule are evaluated for each node of the set. A single assertion is checked by evaluating the assertion expression with the respective node as context. A rule validates a node if all assertions succeed.

A *pattern* is an ordered set of rules. The lexical order of the rule elements in the Schematron document acts as an if-then-else switch for nodes of the instance document. A node is checked with at most one rule per pattern.

A *schema* is an unordered set of patterns. The pattern elements are grouped into a sch:schema element. This element has a queryBinding attribute with an identifier for the language of assertion and context selector expressions.

Originally Schematron was designed as a schema language solely on top of XSLT. The use of XPath and XSLT match expressions was generalized into the concept of a query language in the course of the ISO standardization process. [JELLIFFE2002] [JELLIFFE2003] A query language is defined by a *query language binding* that specifies the query language for rule context expressions and for assertion tests. The initial 2006 specification defined the query language binding for XSLT 1.0 [XSLT1] as the default query language. The later editions added normative query language bindings for XPath 2.0 [XPATH2], XSLT 2.0 [XSLT2], XPath 3.0 [XPATH3], and XSLT 3.0 [XSLT3].

Assertion tests and context expressions are evaluated during validation. They depend on a properly initialized *evaluation environment*. Schematron provides two native elements to set up the environment for query language expressions.

The repeatable sch:ns element is a top-level element that defines the available namespace bindings. Namespace bindings have global scope. The repeatable sch:let element defines the available variables and their values. Variables are globally available or scoped to the containing pattern or rule.

Variables and namespaces can be seen as a common denominator of query languages. The 2016 and 2020 updates to the specification, on the other hand, forbid the use of sch:let for XPath 2.0 and 3.0.¹

Defining namespace bindings and variables is not sufficient to evaluate an XSLT expression. To set up an evaluation environment for XSLT, Schematron explicitly allows the two elements `xsl:key` and `xsl:function` as top-level elements.

2.1.2. Shortcomings

Setting up the query language environment became more elaborate with new functionality introduced with XSLT 2.0 and later 3.0. The 2016 specification addressed this new functionality by allowing the `xsl:function` declaration as an additional top-level element and by restricting the use of the new `fn:error()` function. Following the elimination of the result tree fragment data-type in favor of temporary trees it also made the `value` attribute of the `sch:let` expression optional, allowing temporary trees as value.

The 2020 specification did not add anything in support of XSLT 3.0.

2.1.3. Corrections

With regard to XSLT as query language three pieces are missing.

Support for data types

XSLT 2.0 introduced the capability to declare the required types of, among other things, variables. The type can be a built-in or a user-defined type imported from an XML schema. Schematron is currently missing both: The possibility to declare the required type of a variable and to import user-defined data types.

Setting up accumulators

XSLT 3.0 added accumulators as a new mechanism to capture and process data. An accumulator associates a series of values with nodes of the document tree. The value of an accumulator is obtained by calling the respective accumulator functions `fn:accumulator-before()` and `fn:accumulator-after()`. Although accumulators are designed for streaming transformations they can also be used with non-streaming transformations.

Loading user-defined libraries

The addition of user-defined functions in XSLT 2.0 opened up the possibility of reusable, user-defined function libraries. XSLT 3.0 made the notion of user-defined libraries explicit by adding *packages*, collections of stylesheet modules with a controlled interface.

Schematron lacks a way of including user-defined function libraries other than using schema composition instructions. But neither `sch:include` nor `sch:extends` is a good fit. The former inserts the content of an XML document in place of the element, the latter is only allowed to extend a rule.²

To address these missing pieces the query language specification for XSLT should be expanded as follows:

- ◊ allow `xsl:include` and `xsl:import` as top-level elements for the query languages XSLT 1.0, 2.0, and 3.0
- ◊ allow `xsl:schema-import` as top-level elements for the query languages XSLT 2.0 and 3.0

¹It is unclear why the working group felt the need to forbid the use of `sch:let`. All versions of XPath support variable references in XPath expressions.

²See Section 2.2 [21]

- ◇ allow an optional as attribute on a sch:let instruction
- ◇ allow xsl:use-package and xsl:accumulator as top-level elements for the query language XSLT 3.0

Table 2. Query language environment for XSLT (proposed additions marked with an asterisk)

	XSLT 1.0	XSLT 2.0	XSLT 3.0
Indexes	xsl:key	xsl:key	xsl:key
Functions	-	xsl:function	xsl:function
Namespaces	sch:ns	sch:ns	sch:ns
Variables	sch:let	sch:let	sch:let
User-defined libraries*	xsl:include, xsl:import	xsl:include, xsl:import	xsl:include, xsl:import, xsl:use-package
Typed variables*	-	sch:let/as	sch:let/as
User-defined types*	-	sch:import-schema	sch:import-schema
Accumulators*	-	-	xsl:accumulator

2.2. Schema composition

2.2.1. Summary

Schema composition is about arranging schemas, patterns, rules, and assertions to support different validation tasks and for the ease of maintenance by allowing reuse of assertions, rules, and patterns.

Schematron provides four ways to compose a schema.

An *abstract rule* is a named set of assertions and represented by a sch:rule element without a context expression. It is scoped to the containing pattern. All rules of the same pattern can use a set of assertions by referencing the name of the abstract rule in the rule attribute of a sch:extends element.

A *phase* is a named set of patterns defined by a top-level sch:phase element. Phases allow one schema document to cover multiple validation scenarios or needs.

Schematron offers two mechanisms to use external definitions. The sch:include element can be used anywhere and has a href attribute that points to an external resource. The content of the resource is inserted in place of the sch:include element.

The second mechanism is build on top of the sch:extends element. Instead of a rule attribute naming an abstract rule, it may use a href attribute pointing to an external resource. If the external resource is the same element as the sch:extends element's parent, then the child nodes of the referenced element are inserted in place of the sch:extends element. The use of sch:extends however is still restricted to the sch:rule element.

2.2.2. Shortcomings

Consider the following case. A schema A, shown in Figure 1 [22], includes a pattern from a different schema B, shown in Figure 2 [22].

Figure 1. schema-a.sch

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt2">
  <sch:ns prefix="example" uri="http://example.com/ns"/>
  <sch:include href="schema-b.sch#pattern-b"/>
</sch:schema>
```

Figure 2. schema-b.sch

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt2">
  <sch:ns prefix="ex" uri="http://example.com/ns"/>
  <sch:let name="allowedTypes" value=" tokenize('foo bar', ' ')"/>
  <sch:pattern id="pattern-b">
    <sch:rule context="ex:element[@type = $allowedTypes]">
      <sch:assert test="false()"/>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

Simply replacing the sch:include element in A with the pattern from B does not produce a working schema.

- ◊ The included pattern was written with the expectation that the prefix ex is bound to <http://example.com/ns1>, but schema A uses the prefix example.
- ◊ The rule context expression refers to a global variable allowedTypes which is not defined in A.

In general, the interaction of schema composition with rule based validation is underspecified.

2.2.3. Corrections

There is no simple correction. Schematron requires an inclusion mechanism that covers the interaction between variables and namespaces of combined schemas.

2.3. Reporting

2.3.1. Summary

Schematron encourages the use of natural language descriptions targeted to human readers. This way validation can be more than just a binary distinction (document valid/invalid) but also support authors of in-progress documents with quick feedback on erroneous or unwanted document structure and content.

To this effect an assertion test is associated with zero or more natural-language statements written in a lightweight templating language. The specification distinguishes between a primary statement given in the element content of an assertion element, and secondary statements or *diagnostic* given in the element content of a sch:diagnostic element and referenced in a diagnostics attribute of an assertion element.

The 2016 revision added a mechanism for structured data targeted at machine rather than human consumption. Designed in the same fashion as diagnostics, an assertion may reference zero or more sch:property elements in a properties attribute. Each property element contains a template for structured data.

The templates for natural-language statements and structured data are instantiated when the respective assertion fails. The templating language contains both, elements for text

formatting (`sch:emph`, `sch:span`, `sch:dir`) and elements that calculate values from the instance document (`sch:value-of`, `sch:name`; `xsl:copy-of` in `sch:property` for the query languages XSLT 2.0 and 3.0).

2.3.2. Shortcomings

Figure 3. Multi-lingual schema suggested by Annex G of ISO/IEC 19757-3:2020

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:title>Example of Multi-Lingual Schema</sch:title>
  <sch:pattern>
    <sch:rule context="dog">
      <sch:assert test="bone" diagnostics="bone-en bone-de"/>
    </sch:rule>
  </sch:pattern>
  <sch:diagnostics>
    <sch:diagnostic xml:lang="en" id="bone-en">A dog should have a bone.</sch:diagnostic>
    <sch:diagnostic xml:lang="de" id="bone-de">Ein Hund sollte ein Bein haben.</sch:diagnostic>
  </sch:diagnostics>
</sch:schema>
```

All three reporting elements are modelled as mixed-content elements. As a consequence, an assertion message or diagnostic can only specify one language with an `xml:lang` attribute. The suggested mechanism for multi-lingual messages in Annex G of the specification, shown in Figure 3 [23], is cumbersome, at best. It suggests the schema author to define one diagnostic per localized message and reference them in the `diagnostics` attribute of the respective assertion. Tobias Fischer [FISCHER2017] summarizes the biggest problem with this approach: Adding a localization requires adding the respective reference to every assertion.

2.3.3. Proposed corrections

To address this shortcoming Schematron should make the abstraction of a specific message and its natural language content explicit. Rather than treating localized variants as different diagnostics, they should be treated as one diagnostic with different localized variants. A `sch:message` element that wraps the mixed-content templates would do the trick.

Figure 4. Multi-lingual schema, improved

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:title>Improved Example of Multi-Lingual Schema</sch:title>
  <sch:pattern>
    <sch:rule context="dog">
      <sch:assert test="bone" diagnostics="bone"/>
    </sch:rule>
  </sch:pattern>
  <sch:diagnostics>
    <sch:diagnostic id="bone">
      <sch:message xml:lang="en">A dog should have a bone.</sch:message>
      <sch:message xml:lang="de">Ein Hund sollte ein Bein haben.</sch:message>
    </sch:diagnostic>
  </sch:diagnostics>
</sch:schema>
```

3. From ISO Schematron to Open Schematron?

In the previous sections I discussed some shortcomings of the current ISO Schematron specification. Some could be fixed by simply amending the ISO specification, some like the lacking inclusion mechanism require more substantial work. Ideally, these issues would be

addressed by a future 4th edition of the ISO Schematron specification. But alas, this seems unlikely.

Maybe ISO Schematron becoming proprietary gives us the opportunity to start anew; or rather start with the core ideas of the Schematron assertion language and specify a rule based validation language that is not identical to, but intersects with ISO Schematron. This new Schematron language has the advantage of roughly 15 years of development in markup technology and roughly 15 years of ISO Schematron practice.

It should be based on the XQuery and XPath Data Model 3.1 [WALSH2017], a data model that includes maps, arrays, and support for strongly typed languages with a type system. It should also distinguish between a core specification and language extensions, and describe an extension mechanism that decouples the core from query language or use-case specific extensions such as streaming validation [DZIURLAJ2021] or interactive corrections [KUTSCHERAUER2018]. Given the rise of JSON [JSON] it should try to align with jsontron [AMER2014] and provide a query language binding for JSON-based path languages.

Most importantly, it should again be an Open Standard.

Bibliography

- [AMER2014] Ali, Amer. 2014. Schematron Based Semantic Constraints Specification Framework & Validation Rules Engine for JSON. <https://raw.githubusercontent.com/amer-ali/jsontron/master/docs/Jsontron-presentation-v1.pdf>.
- [XPATH2] Berglund, Anders, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon, eds. 2010. XML Path Language (XPath) 2.0 (Second Edition). World Wide Web Consortium. <http://www.w3.org/TR/2010/REC-xpath20-20101214/>.
- [XSLT1] Clark, James, ed. 1999. XSL Transformations (XSLT) Version 1.0. World Wide Web Consortium. <http://www.w3.org/TR/1999/REC-xslt-19991116>.
- [JSON] Crockford, Douglas. 2017. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259. Internet Engineering Task Force (IETF). <https://tools.ietf.org/pdf/rfc8259.pdf>.
- [DZIURLAJ2021] Dziurlaj, John. 2021. Streaming Extensions for Schematron. <https://github.com/HiltonRoscoe/streamable-schematron>.
- [FISCHER2017] Fischer, Tobias. 2017. Localization Concept Needs Improvement. <https://github.com/Schematron/schematron/issues/40>.
- [JELLIFFE1999] Jelliffe, Rick. 1999. Schematron: An Interview with Rick Jelliffe Interview by Simon St.Laurent. <https://web.archive.org/web/20010619150155/http://www.xmlhack.com/read.php?item=121>.
- [JELLIFFE2002] Jelliffe, Rick. 2002. The Schematron Assertion Language 1.6. <https://web.archive.org/web/20061230150144/http://xml.ascc.net:80/resource/schematron/Schematron2000.html>.
- [JELLIFFE2003] Jelliffe, Rick. 2003. Schematron 1.5 to Schematron 1.6 to ISO Schematron. June 21, 2003. <http://www.topologi.com/resources/schematronUpgrades.html>.

- [KUTSCHERAUER2018] Kutscherauer, Nico, and Octavian Nadolu, eds. 2018. Schematron Quick Fixes Specification. Quick-fix support for XML Community Group. <http://schematron-quickfix.github.io/sqf/spec/SQFSpec.html>.
- [XSLT3] Kay, Michael, ed. 2017. XSL Transformations (XSLT) Version 3.0. World Wide Web Consortium. <https://www.w3.org/TR/2017/REC-xslt-30-20170608/>.
- [XSLT2] Kay, Michael, ed. 2021. XSL Transformations (XSLT) Version 2.0 (Second Edition). World Wide Web Consortium. <https://www.w3.org/TR/2021/REC-xslt20-20210330/>.
- [XPATH3] Robie, Jonathan, Don Chamberlin, Micheal Dyck, and John Snelson, eds. 2014. XML Path Language (XPath) 3.0. World Wide Web Consortium. <http://www.w3.org/TR/2014/REC-xpath-30-20140408/>.
- [SCHEMATRON2020] Information technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based validation, Schematron, second edition, International Standard ISO/IEC 19757-3:2020, Geneva, Switzerland : ISO
- [WALSH2017] Walsh, Norman, John Snelson, and Andrew Coleman, eds. 2017. XQuery and XPath Data Model 3.1. World Wide Web Consortium. <https://www.w3.org/TR/2017/REC-xpath-datamodel-31-20170321/>.



XSLTForms for the '20s

Alain Couthures, agenceXML

Since 2009, XSLTForms has been developed with support for old browsers. It is now time to consider modern browsers and XForms 2.0 support with XPath/XQuery 2.0+ instead of XPath 1.0.

More and more complex forms are to be used but do they all work as users would like? XForms is remaining as a nice declarative approach to allow authors, even without strong programming abilities, to write robust forms. XSLTForms is a light XForms implementation for browsers, at client side.

1. XSLTForms since 2009

XSLTForms [XSLTForms] has been compatible with successive browser versions starting from Internet Explorer 4. It is based on XSLT 1.0, CSS and vanilla Javascript. Consequently, XSLTForms is composed of 3 files: `xsltforms.xsl`, `xsltforms.css` and `xsltforms.js`.

This XSLT stylesheet acts as a compiler, generating both HTML elements with a bunch of CSS classes and Javascript instructions. The complex part in it is the XPath 1.0 parser.

For example,

```
<xf:input ref="PersonGivenName" incremental="true">
  <xf:label>Please enter your first name: </xf:label>
</xf:input>
```

Becomes

```
<span id="xsltforms-mainform-input-4_4_2_" class="xforms-disabled xforms-control xforms-input xforms-appearance">
  <span class="focus"> </span>
  <label for="xsltforms-mainform-input-input-4_4_2_" id="xsltforms-mainform-label-2_4_4_2_" class="xforms-label">Please enter your first name: </label>
  <span class="value">
    <input type="text" id="xsltforms-mainform-input-input-4_4_2_" incremental="true" class="xforms-value" />
  </span>
  <span class="xforms-required-icon">*</span>
  <span class="xforms-alert">
    <span class="xforms-alert-icon"> </span>
  </span>
</span>
```

And

```
XsltForms_xpath.create(xsltforms_subform, "PersonGivenName", false,
  new XsltForms_locationExpr(false,
    new XsltForms_stepExpr('child',
      new XsltForms_nodeTestName('', 'PersonGivenName')
    )
  )
);
new XsltForms_input(xsltforms_subform, xsltforms_subform.id + "-input-4_4_2_", 2, "text",
```

```
new XsltForms_binding(null, "PersonGivenName"),null,"true",null,null,null
);
```

XForms 1.1 Recommendation [XF11] is based on XPath 1.0: no sequences, no eq, ne, lt, ... operators, no types, ... XSLTForms has its own XPath 1.0 engine in two steps: the parser written in XSLT 1.0 and the evaluator written in Javascript. It is frustrating for authors used to retrieve data from XML databases to be restricted to XPath 1.0 while XQuery/XPath 4.0 is now emerging.

For example,

```
concat('Hello ', PersonGivenName, '. We hope you like XForms!')
```

Becomes

```
new XsltForms_functionCallExpr('http://www.w3.org/2005/xpath-functions concat',
  new XsltForms_cteExpr('Hello '),
  new XsltForms_locationExpr(false,
    new XsltForms_stepExpr('child',
      new XsltForms_nodeTestName('', 'PersonGivenName')
    )
  ),
  new XsltForms_cteExpr('. We hope you like XForms!')
)
```

For generating Javascript instructions, XSLT 1.0 is not a very nice approach because of a rather limited function set and just named templates to process strings. For example, performance is not good with very long element names... When applied at client-side, the XSLT stylesheet execution results in a blank-page effect before HTML elements being rendered.

2. Browsers have evolved

XSLT 1.0 is still available on major browsers but some smartphones don't have it. Some browser vendors would be happy removing their XSLT engine just to gain memory. They usually argue that XSLT 1.0 is not significantly used in websites. And Mozilla Firefox own XSLT 1.0 engine bugs won't be fixed...

HTML, CSS and Javascript have evolved significantly to HTML5, CSS3 and ECMAScript 6 and browsers support many of their features which allows more powerful programming.

Machines are faster and Javascript is more and more optimized for performance.

Vanilla Javascript can still be a pertinent choice because even once famous Javascript frameworks come and go while others don't bother with ascending compatibility... They do not seem to be really interested in performance either.

By the way, Node.js appeared so vanilla Javascript can also run outside a browser. Just using it, it is very easy to run an HTTP server on a local machine: Node.js can be used to interact with files, databases and so on while XForms is interacting with users.

All these evolutions allow XSLTForms modernization to be declined in the following axes:

- ◇ An HTML5 notation for XForms elements and CSS styling of them
- ◇ An XQuery/XPath 3.1 parser written in Javascript

- ◊ An XQueryX 3.1 to Javascript transpiler

XForms 2.0 [XF2] is also in good progress and XSLTForms should support all its features as well.

3. HTML5 notation for XForms elements and CSS styling

XSLTForms is now using an HTML5 notation for XForms elements replacing numerous DIV and SPAN embedded elements with only CSS classes to differentiate them. This is still performed with XSLT 1.0 but it can also be done with vanilla Javascript and Node.js.

HTML5 is allowing custom elements and CSS classes can be defined for them. To avoid name conflicts, XForms elements are transformed into xforms-* elements (xforms-input, for example) and XForms attributes into xf-* attributes (xf-selected, for example). Shadow DOM is not currently used in XSLTForms but it cannot be used for grouping controls anyway.

Each HTML attribute with AVT is renamed with the prefix xf-template- and the xf-avt attribute is added to the corresponding HTML element.

XForms repeat elements are automatically converted into XForms repeat attributes when in HTML tables.

For example,

```
<xf:input ref="PersonGivenName" incremental="true">
    <xf:label>Please enter your first name: </xf:label>
</xf:input>
```

Becomes

```
<xforms-input xf-ref="PersonGivenName" xf-incremental="true">
    <xforms-label>Please enter your first name: </xforms-label>
</xforms-input>
```

There is no Javascript instructions generation in the new XSLT stylesheet.

HTML5 is not allowing custom elements within HEAD element so xforms-model elements have to go within BODY element.

Inline XML instances and XML schemas have to be protected within SCRIPT elements with a not-supported type (application/xml, for example) to preserve letter cases. Because of a design error in Mozilla Firefox XSLT engine, instances are also to be serialized with entities.

For example,

```
<xf:instance>
    <data xmlns="">
        <PersonGivenName/>
    </data>
</xf:instance>
```

Becomes

```
<xforms-instance>
    <script type="application/xml">&lt;data xmlns=""&gt;&lt;PersonGivenName/&gt;&lt;/data&gt;</script>
</xforms-instance>
```

As a consequence, for XSLTForms, the XSLT 1.0 stylesheet is lighter and the transformation is more than 3 times faster which is nicer for users, reducing the blank page wait.

The XSLT stylesheet is much simpler and it becomes easy to rewrite it in a procedural programming language. It could also be done with XQuery [XQuery] [XPF], of course. It has, actually, already been done in Javascript for Node.js because browser debuggers didn't allow anymore breakpoints in Javascript when resulting from an XSLT transformation (it has been fixed in browsers since, but a few months later).

It is also possible to directly write forms in HTML5 as the XSLT stylesheet would generate them. When generating XForms pages with XQuery at server side, it might be more convenient than finding some trick to add the XSLT processing instruction to be applied at client side not at server-side. It is, of course, also good for performance and there is no blank page effect at all.

Yet, XSLTForms requires more custom sub elements, such as `xforms-body`, `xforms-help`, ... to render XForms controls with HTML and CSS. This is now performed with Javascript when building the form.

For example,

```
<xforms-input xf-ref="PersonGivenName" xf-incremental="true">
  <xforms-label>Please enter your first name: </xforms-label>
</xforms-input>
```

Becomes

```
<xforms-input xf-ref="PersonGivenName" xf-incremental="true">
  <xforms-label>Please enter your first name: </xforms-label>
  <xforms-body>
    <input>
  </xforms-body>
  <xforms-required></xforms-required>
  <xforms-alert-mark></xforms-alert-mark>
  <xforms-alert></xforms-alert>
</xforms-input>
```

Because of CSS attribute selectors, CSS rules are powerful enough to manage custom elements visibility. XSLTForms is not anymore adding its own CSS classes: it is adding run-time attributes instead. For example, an XForms group without a binding will always be visible but, with a binding, an extra attribute (`xf-bound`) will be programmatically added when effectively bound to enable visibility. Models and actions are always hidden.

Default CSS rules have been aligned to CSS rendering of XForms tutorial written by Steven Pemberton [XFTUT].

The `xsltforms.css` file has, effectively, been fully rewritten.

For example,

```
xforms-input:not([xf-required])[xf-invalid] > xforms-label::after {
  content: "\2716";
  color: red;
}
```

4. XQuery/XPath parser in Javascript

XForms is evaluating most XPath expressions in forms repeatedly. So, it is much more efficient, especially in browsers, to parse them once and preserve some compiled form for repeated evaluations.

Previously, XSLTForms was using XSLT to parse XPath 1.0 expressions into Javascript code to create objects with a `.run()` method. Performance was good enough but with some unexpected delays for very long node names in XML instances...

This XPath parser has been rewritten in Javascript to be executed just-in-time when an expression is to be evaluated for the first time. Again, the XSLT stylesheet is lighter and the blank page wait is reduced.

Because the previous parser was targeting XPath 1.0, the new one has been extended to support XQuery 3.1 and generate XQueryX [XQueryX] stored in Javascript arrays (each element or attribute as an array of name and value).

For example,

```
concat('Hello ', PersonGivenName, '. We hope you like XForms!')
```

Becomes

```
[Fleur.XQueryX.functionCallExpr, [
  [Fleur.XQueryX.functionName, ['concat']],
  [Fleur.XQueryX.arguments, [
    [Fleur.XQueryX.stringConstantExpr, [[Fleur.XQueryX.value, ['Hello ']]]],
    [Fleur.XQueryX.pathExpr, [
      [Fleur.XQueryX.stepExpr, [
        [Fleur.XQueryX.xpathAxis, ['child']],
        [Fleur.XQueryX.nameTest, ['PersonGivenName']]
      ]]
    ]],
    [Fleur.XQueryX.stringConstantExpr, [[Fleur.XQueryX.value, ['. We hope you like XForms!']]]]
  ]]
]]
```

Which can be serialized in XQueryX as

```
<xqx:functionCallExpr>
  <xqx:functionName>concat</xqx:functionName>
  <xqx:arguments>
    <xqx:stringConstantExpr>
      <xqx:value>Hello </xqx:value>
    </xqx:stringConstantExpr>
    <xqx:pathExpr>
      <xqx:stepExpr>
        <xqx>xpathAxis>child</xqx>xpathAxis>
        <xqx:nameTest>PersonGivenName</xqx:nameTest>
      </xqx:stepExpr>
    </xqx:pathExpr>
    <xqx:stringConstantExpr>
      <xqx:value>. We hope you like XForms!</xqx:value>
    </xqx:stringConstantExpr>
  </xqx:arguments>
</xqx:functionCallExpr>
```

On Node.js, the XQueryX structure is, then, the input for a new XQuery engine, named Fleur [Fleur], which also includes a DOM3 engine. It has to run asynchronous

calls on functions such as `fn:doc()`, `http:send-request()`, `file:write()`, `prof:sleep()`...

Because `async/await` feature was not yet being supported in Node.js and old browsers, current version Fleur is still struggling with callbacks hell, call stack overflow. Using this version from XSLTForms would have needed a massive reorganization of sources: a converter from XQueryX to old XPath 1.0 objects of XSLTForms has, temporarily, been added to remove the previous XPath 1.0 parser written in XSLT 1.0.

The `fleur()` Javascript function has been added to be used within the Console in the browser debugger to evaluate XPath expressions.

For example,

```
fleur("instance()")
```

Returns

```
"<data xmlns=\"\"><PersonGivenName/></data>"
```

Because rendering in Javascript console is rather limited, an XForms console has been created (accessible with F1 key). Currently limited to expression evaluation, it will be enriched to render, with HTML, Model Item Properties (`type`, `relevant`, `required`, `valid`, `read-only`,...) bound to XForms instance data nodes.

With Node.js easily allowing to run an http server, Fleur is also a nice XQuery Web server for XSLTForms to generate inline XML instances, manipulate files, REST APIs,...

5. XQueryX 3.1 to Javascript transpiler

Now, all browsers support `async/await` feature, so does Node.js. The Javascript arrays resulting from the XPath/XQuery parser can now be converted into Javascript functions.

To evaluate an expression, the context is passed as unique parameter to the Javascript function. Its body contains a sequence of instructions with a Reverse Polish Notation order: arguments and parameters are stacked before function calls. Path evaluations require `if` statements to stop when a step returns an empty sequence. Predicates are performed with inline functions to be called for each sequence item. Error management is implemented throwing Javascript exceptions.

For example,

```
concat('Hello ', PersonGivenName, '. We hope you like XForms!')
```

Becomes

```
ctx => {
  ctx.xqx_stringConstantExpr('Hello ');
  ctx.xqx_pathExpr();
  ctx.xqx_xpathAxis_child();
  ctx.xqx_nameTest("PersonGivenName");
  ctx.restoreContext();
  ctx.xs_string_1();
  ctx.xqx_stringConstantExpr('. We hope you like XForms!');
  ctx.fn_concat(3);
  return ctx;
};
```

This is optimized for Javascript interpreter own call stack and `async/await` use is reduced to minimal.

For example,

```
doc('r2d2.urdf.xml')/robot/joint[@name eq 'swivel']
```

Becomes

```
async ctx => {
    ctx.xqx_pathExpr();
    await ctx.xqx_filterExpr_async(async ctx => {
        ctx.xqx_stringConstantExpr('r2d2.urdf.xml');
        await ctx.fn_doc_1_async();
        return ctx;
    });
    if (ctx.item.isNotEmpty()) {
        ctx.xqx_xpathAxis_child();
        ctx.xqx_nameTest("robot");
        if (ctx.item.isNotEmpty()) {
            ctx.xqx_xpathAxis_child();
            ctx.xqx_nameTest("joint");
            if (ctx.item.isNotEmpty()) {
                ctx.xqx_predicateExpr(ctx => {
                    ctx.xqx_pathExpr();
                    ctx.xqx_xpathAxis_attribute();
                    ctx.xqx_nameTest("name");
                    ctx.restoreContext();
                    ctx.atomize();
                    ctx.xqx_stringConstantExpr('swivel');
                    ctx.xqx_valueComp(Fleur.eqOp);
                    return ctx;
                });
            }
        }
    }
    ctx.restoreContext();
    return ctx;
};
```

It is also enabling static optimization when generating the Javascript source. For example, atomizing and node sorting are not always necessary when calling a function. Instead of Javascript, with XSLT, a more sophisticated transpiler could be written to transform XQueryX to Javascript functions.

The `fleur()` Javascript function returns a type-explicit serialization which simulates an equivalent XQuery expression for the result of evaluation (more explicit than adaptative output).

For example,

```
fleur("xs:date('2021-05-21') - xs:date('2021-05-10'))
```

Returns

```
"xs:dayTimeDuration('P11D')"
```

6. XForms 2.0 support in XSLTForms

XForms 2.0 comes with XPath 2.0 and later support. XSLTForms will have XPath 3.1 soon...

XSLTForms is already supporting some major XForms 2.0 new features such as variables and AVT.

XForms 2.0 Test Suite can be checked with XSLTForms running it. As for XForms 1.1 Test Suite, the corresponding form will be extended to produce the new Test Report.

New extensions will also be considered such as the use of HTML5 TEMPLATE elements to populate `xforms-body` elements, or the ability to script XForms actions and use XQuery Update Facility [XQUF] when applicable.

Bibliography

[XSLTForms] Couthures, Alain: XSLTForms. <https://github.com/AlainCouthures/declarative4all/tree/master/public/direct>

[Fleur] Couthures, Alain: Fleur. <https://github.com/AlainCouthures/declarative4all/blob/master/build/js/fleur.js>

[XQuery] Robie, Jonathan - Dyck, Michael (eds.): XQuery 3.1: An XML Query Language. W3C, 2017. <http://www.w3.org/TR/xquery-31/>

[XPF] Kay, Michael (ed.): XPath and XQuery functions and operators 3.1. W3C, 2017. <https://www.w3.org/TR>xpath-functions-31/>

[XQUF] Snelson, John - Melton, Jim (eds.): XQuery Update Facility 3.0. W3C, 2015. <http://www.w3.org/TR/xquery-update-30/>

[XQueryX] Melton, Jim - Spiegel, Josh (eds.): XQueryX 3.1. W3C, 2017. <http://www.w3.org/TR/xquery-31/>

[XF11] Boyer, John M. (ed.): XForms 1.1. W3C, 2009. <https://www.w3.org/TR/2009/REC-xforms-20091020/>

[XF2] Bruchez, Eric - al. (eds.): XForms 2.0. W3C, 2018. https://www.w3.org/community/xformsusers/wiki/XForms_2.0

[XFTUT] Pemberton, Steven: XForms Hands On Tutorial. 2020. <https://homepages.cwi.nl/~steven/xforms/xforms-hands-on/tutorial.xhtml>



<transpile from="Java" to="C#" via="XML" with="XSLT"/>

Michael Kay, Saxonica

This paper describes a project in which XSLT 3.0 was used to convert a substantial body of Java code (around 500K lines of code) to C#. The Java code, as it happens, is the source code of the Saxon XSLT processor, but that's not really relevant: it could have been anything.

1. Introduction

For a number of years, Saxonica has developed the Saxon product¹, a Java implementation of the W3C XSLT, XQuery, XPath, and XSD specifications. The product has also been made available on the .NET platform, by converting the bytecode generated by the Java compiler into the equivalent intermediate language (called IL) on .NET. The tool for this conversion was the open-source IKVMC library² developed by Jeroen Frijters.

IKVMC was largely a one-man project, and when Jeroen (after many years of faithful service to the community) decided to move on to other things, there was no-one to step into his capable shoes, and the project has languished.

In 2019, Microsoft announced a change of direction for the .NET platform³. .NET had diverged into two separate strands of development, known as .NET Framework and .NET Core, and Microsoft announced in effect that .NET Framework would be discontinued, and the future lay with .NET Core. The differences between the two strands need not really concern us here, except to note that IKVMC never supported .NET Core, therefore Saxon didn't run on .NET Core, and therefore we needed to find a different way forward.

The way that we chose was source code conversion from Java to C#. At the time of writing this has been successfully achieved for a large subset of the Saxon product, and work is ongoing to convert the remainder. This paper describes how it was done.

Let's start by describing the objectives of the project:

- ◊ Automated conversion of as much of the source code as possible from Java to C#.
- ◊ Repeatable conversion: this is not a one-off conversion to create a fork of the code; we want to continue developing and maintaining the master Java code and port all changes over to C# using the same conversion technology.
- ◊ Performance: the performance of the final product on .NET must be at least as good as the existing product. In fact, we would like it to be considerably better, because (for

¹<http://www.saxonica.com/>

²<http://www.ikvm.net/>

³<https://devblogs.microsoft.com/dotnet/net-core-is-the-future-of-net/>

reasons we have never fully understood) some workloads on the current product perform much more slowly than on the Java platform.

- ◊ Maintainability: although we don't intend to develop the C# code independently, we will certainly need to debug it, and that means we need to generate human-readable code.
- ◊ Adaptability: because the .NET platform is different from the Java platform, some parts of the product need to behave differently. We need to have convenient mechanisms to manage these differences.

I should also stress one non-objective: we were not setting out to provide a tool that could convert any Java program to C# fully automatically. We only needed to convert one (admittedly rather large) program, and this meant that:

1. We only needed to convert Java constructs that Saxon actually uses (which turns out to be quite a small subset of the total Java platform).
2. In the case of constructs that Saxon uses rarely, we could do some manual assistance of the conversion, rather than requiring it fully automatic. Indeed, by Zipf's law, many of the Java constructs that Saxon uses are only used once in the entire product, and in many cases they are used unnecessarily and could easily be rewritten a different way (sometimes beneficially). The main device we have used for this manual assistance is the use of Java annotations in the source code, annotations that are specially recognised as hints by the converter.

2. Preliminaries

Our initial investigations explored a number of available tools for source code conversion. The only one that looked at all promising was a commercial product, Tangible⁴. We bought a license to evaluate its capabilities, and the exercise taught us a lot about where the difficulties were going to arise. It was immediately apparent that we would have considerable difficulties with Java generics, with anonymous inner classes, and with our extensive use of the Java CharSequence interface, which has no direct equivalent in .NET. The exercise also taught us that Tangible, on its own, wasn't up to the job. (Having said that, the conversions performed by Tangible helped us greatly in defining our own rules.)

Our next step was to reduce our dependence on the constructs that were going to prove difficult to convert: especially generics, and the use of CharSequence. I have described in more detail how we achieved this in blog postings: ⁵ ⁶.

Generics are difficult because although Java and C# use superficially-similar syntax (for example `List<String>`) the semantics are very different. In C# instances are annotated at run-time with the full expanded type, and one can therefore write run-time tests such as `x is List<String>`. Writing `x is List` will return false: `List<String>` is not a subtype of `List`. By contrast, with Java, the type parameters are used only at compile time and are discarded at run time (the process is called *Type Erasure*). This means that on Java, `x instanceof List<String>` is not allowed, while `x instanceof List` returns true.

We decided to reduce the scale of the problem by dropping some of our use of generics from the product. In particular, in Saxon 9.9, two key interfaces, `Sequence` and `SequenceIterator`, were defined with type parameters restricting the type of items contained in an XDM sequence, and we dropped this in Saxon 10.0. The use of type parameters here had always been somewhat unsatisfactory, for two reasons:

⁴<https://www.tangiblesoftwareresolutions.com>

⁵<https://dev.saxonica.com/blog/mike/2020/07/string-charssequence-ikvm-and-net.html>

⁶<https://dev.saxonica.com/blog/mike/2020/01/java-generics-revisited.html>

Most of the time, the code has to deal with sequences-of-anything: we don't know statically, when we write the Saxon code, what type of input it is going to be dealing with (that depends on the user-written stylesheet). So providing the type parameter (`Sequence<Item>`) simply doesn't add any value.

The XDM model for sequences has the property that an item is a sequence of length one. So `Item` implements `Sequence<Item>`. Which means that a subclass of `Item`, such as `DateTimeValue`, implements `Sequence<DateTimeValue>`. Which followed to its logical conclusion means that a `DateTimeValue` is an `Item<DateTimeValue>`, and a generic item is therefore an `Item<Item>` (or is it an `Item<Item<Item<...>>>?`). Modelling the XDM structure accurately using Java generics proved very difficult, and in the end, it introduced a whole load of complexity without adding much value. Getting rid of it was welcome.

As far as the `CharSequence` interface is concerned, we used this extensively in interfaces where strings are passed around, to enable us to use implementations of strings other than the Java `String` class. For example, the whitespace that often occurs between elements in an XML document is compressed using run-length encoding as a `CompressedWhitespace` object, which implements the `CharSequence` interface, and can therefore be substituted in many cases for a Java `String`.

The use of `CharSequence` isn't perfect for this purpose, however. Firstly, it has the same problem as a `Java String` in that it models a string as a sequence of 16-bit UTF-16 char values, using a surrogate pair to represent Unicode astral codepoints. In XPath, strings need to be codepoint-addressable (at least for the purposes of functions such as `substring()` and `translate()`), and neither `String` nor `CharSequence` meets this requirement. There are also issues concerning comparison across different implementations of the `CharSequence` interface, plus the fact that many commonly used methods in the standard Java class library require the `CharSequence` to be converted to a `String`, which generally involves copying the content. In addition, the `CharSequence` interface doesn't guarantee immutability. For these reasons, we had already introduced another string representation, the `UnicodeString`, which we were using in many corners of the code, notably when processing regular expressions.

C# has no direct equivalent of `CharSequence`: that is, an interface which is implemented by the standard `String` class, but which also allows for other implementations. The interface `IEnumerable<Char>` comes close, but that doesn't allow for direct addressing to get the `N`th character in a string.

So we decided to scrap our extensive use of `CharSequence` throughout the product, and replace it with our own `UnicodeString` interface – which allows for direct codepoint addressing, rather than char addressing with surrogate pairs. There is a performance hit in doing this, because there's a lot of conversion between `String` and `UnicodeString` when data crosses the boundary between Saxon and third-party software (notably the XML parser, but also library routines such as `upperCase()` and `lowerCase()`). However, it's sufficiently small that most users won't notice the difference, and we can mitigate it – for example we have our own UTF-8 Writer used by the Saxon serializer, and it was easy to extend the UTF-8 Writer to accept a `UnicodeString` as input, bypassing the conversion of `UnicodeString` to `String` prior to UTF-8 encoding.

3. Examples of Converted Code

To set the scene, it might be useful to provide a couple of examples of converted code, illustrating the challenges.

Here's a rather simple method in Java:

```
@Override
public AtomicSequence atomize() throws XPathException {
    return tree.getTypedValueOfElement(this);
}
```

And here is the C# code that we generate:

```
public override net.sf.saxon.om.AtomicSequence atomize() {
    return tree.getTypedValueOfElement(this);
}
```

Nothing very remarkable there, but it actually requires a fair bit of analysis of the Java code to establish that the conversion in this case is fairly trivial. For example:

- ◊ The class name `AtomicSequence` has been expanded; this requires analysis of the `import` declarations in the module, and it can't be done without knowing the full set of packages and classes available.
- ◊ The `@Override` declaration is optional in Java, but `optional` is mandatory in C#; moreover they don't mean quite the same thing, for example when overriding methods defined in an interface or abstract class.
- ◊ The conversion of Java `this` to C# `this` works here, but there are other contexts where it doesn't work.

Now let's take a more complex example. Consider the following Java code:

```
public Map<String, Sequence> getDefaultOptions() {
    Map<String, Sequence> result = new HashMap<>();
    for (Map.Entry<String, Sequence> entry : defaultValues.entrySet()) {
        result.put(entry.getKey(), entry.getValue());
    }
    return result;
}
```

In C# this becomes (with abbreviated namespace qualifiers, for readability):

```
public S.C.G.IDictionary<string, n.s.s.o.Sequence> getDefaultOptions() {
    S.C.G.IDictionary<string, n.s.s.o.Sequence> result =
        new S.C.G.Dictionary<string, n.s.s.o.Sequence>();
    foreach (S.C.G.KeyValuePair<string, n.s.s.o.Sequence> entry in defaultValues) {
        result[entry.Key] = entry.Value;
    }
    return result;
}
```

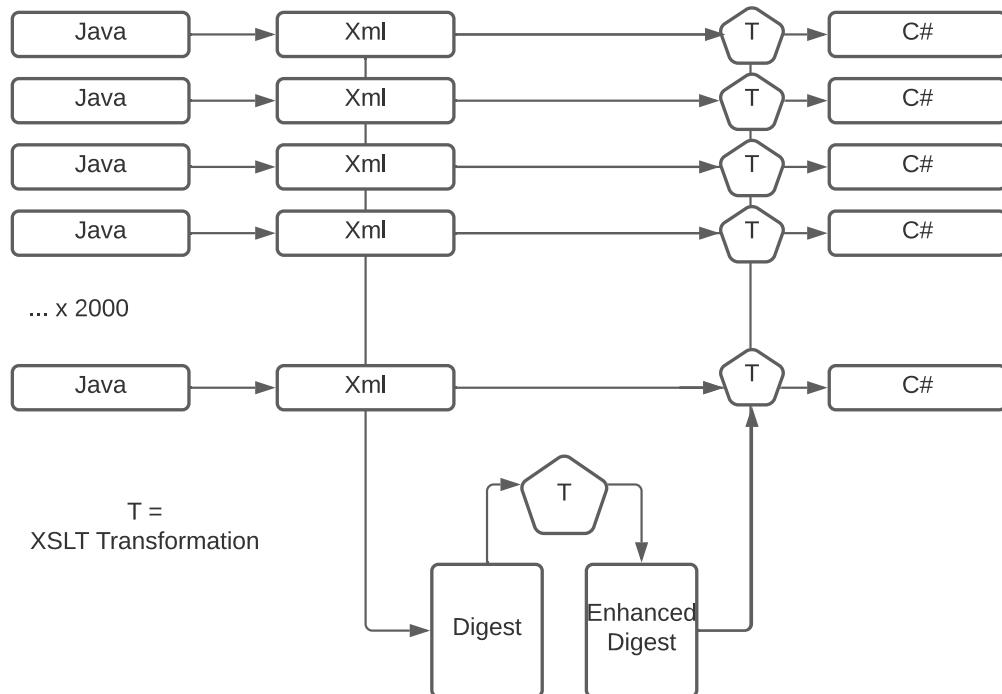
There's a lot going on here:

- ◊ We've replaced the Java `Map` with a C# `Dictionary`, and its `put()` method has been replaced with an indexed assignment;
- ◊ The Java iterable `defaultValues.entrySet()` has been replaced with the C# enumerable `defaultValues`;
- ◊ The references to `entry.getKey()` and `entry.getValue()` have been replaced with property accessors `entry.Key` and `entry.Value`.

- ◇ The replacement of `result.put(key, value)` by `result[key] = value` is fine in this context, but it needs care, because if the return value of the expression is used, the Java code returns the previous value associated with the key, while the C# code returns the new value. The rewrite works here only because the expression appears in a context where its result is discarded.

4. Architecture of the Converter

The overall structure of the transpiler is shown below:



Let's explain this:

- ◇ On the left, we have 2000+ Java modules.
 - ◇ These are converted to an XML representation by applying the JavaParser, and serializing the resulting parse tree as XML.
 - ◇ An XSLT transformation takes all the XML files as input and generates a digest of the class and method hierarchy.
 - ◇ A further XSLT transformation enhances the digest by analyzing which methods override each other.
 - ◇ Each of the 2000+ XML modules is then converted to C# by applying an XSLT transformation, which takes the enhanced digest file as an additional input.

The first stage of conversion is to parse each Java module and generate an abstract syntax tree, which can be serialized as XML. For this purpose we use the open-source JavaParser product⁷.

JavaParser generates the parse tree as a hierarchy of (not very well documented) Java objects. It also includes the capability to serialize this hierarchy as XML. We don't use its

⁷<http://javaparser.org>

out-of-the-box serialization however: we augment it with additional semantic information. JavaParser in fact has two parts (originally developed independently, and still showing evidence of the fact): the parser itself, which is exactly what it says, and the "symbol solver", which is a set of queries that can be executed on the parse tree to obtain additional information. For example, if the raw source code contains the expression new `HashMap<>()`, this will appear in the raw tree as:

```
<value nodeType="ObjectCreationExpr">
  <type nodeType="ClassOrInterfaceType">
    <name nodeType="SimpleName" identifier="HashMap"/>
    <typeArguments/>
  </type>
</value>
```

But with the aid of the symbol solver, it is straightforward to establish that the name `HashMap` refers to the class `java.util.HashMap`, and we output this as an additional attribute on the tree, thus:

```
<value nodeType="ObjectCreationExpr">
  <type nodeType="ClassOrInterfaceType"
        RESOLVED_TYPE="java.util.HashMap">
    <name nodeType="SimpleName" identifier="HashMap"/>
    <typeArguments/>
  </type>
</value>
```

Similarly, the symbol solver is usually able to find the declaration corresponding to a variable reference or method call, and hence to establish the static type of the variable or of the method result. I say usually, because there are cases it gives up on. It struggles, for example, with the types of the arguments to a lambda expression, for example the variable `n` in

```
search.setPredicate(n -> n.name="John")
```

Similarly it has difficulty with static wildcard imports:

```
import static org.w3.dom.Node.*;
```

The other problem with the symbol solver is that it can do a lot of things that aren't mentioned in the documentation: we've found some of these by experiment, or by studying the source code. No doubt there are other gems that remain hidden.

The result of this process is that for each Java module in the product, we generate a corresponding XML file containing its decorated syntax tree.

In principle we could now write an XSLT transformation that serializes this syntax tree using C# syntax. But there's another step first. In some cases we can't generate the C# one file at a time: we need some global information. For example, if a C# method is to be overridden in a subclass, it needs to be flagged with the `virtual` modifier. Similarly, overriding methods need to be flagged as `override`. We therefore need to construct a map of the entire class hierarchy, working out which methods are overridden and which are overrides.

So the second phase of processing is to scan the entire collection of XML documents and generate a digest file (itself an XML document, naturally) which acts as an index of classes, interfaces, and methods, and which represents the class hierarchy of the application. Then

(our third phase) we do a transformation on the digest file which augments it with decisions about which methods are overriding and which are virtual.

Now finally we can perform the XML-to-C# phase, implemented as an XSLT transformation applied to each of the XML documents generated in phase one, but with the digest file available as additional information.

The C# is then ready to be compiled and executed.

5. Difficulties

In this section we outline some of the features of the Java language where conversion posed particular challenges, and explain briefly how these were tackled.

It's worth noting that there are broadly three classes of solution for each of these difficulties:

- ◊ Create an automated conversion that handles the Java construct and converts it to equivalent C#. Note that although this is an automatic conversion, it doesn't necessarily have to handle every edge case, in the way that a productised converter might be expected to do. In particular, it doesn't have to handle edge cases that the Saxon code doesn't rely on: for example the converted code doesn't have to handle `null` as an input in exactly the same way as the original Java, if Saxon never supplies `null` as the input.
- ◊ Convert with the aid of annotations manually added to the Java code. We'll see examples of some of these annotations later.
- ◊ Eliminate use of the problematic construct from the Java code, replacing it with something that can be more easily converted. A trivial example of this relates to the use of names. Java allows a field and a method in a class to have the same name; C# does not. Simple solution: manually rename fields where necessary so that no Saxon class ever has a field and a method with the same name. (Very often, imposing such a rule actually improves the Java code.)

The following sections describe some of the difficulties, in no particular order.

5.1. Dependencies

Java has a class `java.util.HashMap` (which Saxon uses extensively). C# does not have a class with this name. It does have a rather similar class `System.Dictionary`, but there are differences in behavior.

Broadly speaking, there are three ways we deal with dependencies:

- ◊ *Rewriting*. Here the converter (specifically, the XML-to-C# transformation stylesheet) has logic to rename references to the class `java.util.HashMap` so they instead refer to `System.Collections.Generic.Dictionary`, and to convert calls on the methods of `java.util.HashMap` so they instead call the corresponding methods of `System.Dictionary`. We've already seen an example of this above. Sometimes there is no direct equivalent for a particular method, in which case we instead generate a call on a helper method that emulates the required functionality. (`System.Collections.Generic.Dictionary`, for example, has no direct equivalent to the `get()` method on `java.util.HashMap`, largely because it cannot use `null` as a return value when the required key is absent.)

The converter uses rewriting for the vast majority of calls on commonly used classes and methods. There's more detail on how this is done below.

- ◇ *Emulation.* Here we implement (in C#) a class that emulates the behaviour of the Java class – or at least, those parts of the behaviour that Saxon relies on. An example where we do this is `java.util.Properties`, which has no direct equivalent in C#, but which is easily implemented using dictionaries. Saxon doesn't use the complicated methods for importing and exporting `Properties` objects, so we don't need to emulate those.
- ◇ *Avoidance.* Here we simply eliminate the dependency. For example, the Java product will accept input from either a push (SAX) or pull (StAX) parser. On C# we will only support a single XML parser, the one from Microsoft. This is a pull parser, so we eliminate all the Saxon code that's specific to SAX support. This is non-trivial, of course, because the relevant code is widely scattered around the product. But once found, it's usually easy to get rid of it using preprocessor directives in the Java (`//#if CSHARP==false`). I should perhaps have mentioned that there's a "phase 0" in our conversion pipeline, which is to apply these preprocessor directives.

In cases where dependencies are handled by rewriting, there are two parts to this. Firstly, we have a simple mapping of class names. This includes both system classes and Saxon-specific classes. Here are a few of them:

```
<xsl:variable name="specialTypes"
    as="map(xs:string, xs:string)"
    select="map{
'boolean':                 'bool',
'java.io.BufferedInputStream':           'System.IO.BufferedStream',
'java.io.BufferedOutputStream':           'System.IO.BufferedStream',
'java.io.BufferedReader':                'Saxon.Impl.Helpers.BufferedReader',
'java.lang.ArithmeticsException':       'System.ArithmeticsException',
'java.lang.ArrayIndexOutOfBoundsException': 'System.IndexOutOfRangeException',
'java.lang.Boolean':                   'System.Boolean',
'java.lang.Byte':                     'System.Byte',
...
'java.math.BigDecimal':                'Singulink.Numerics.BigDecimal',
...
'java.util.ArrayList':                 'System.Collections.Generic.List',
'java.util.Collection':               'System.Collections.Generic.ICollection',
'java.util.Comparator':                'System.Collections.Generic.Comparer',
...
'net.sf.saxon.ma.trie.ImmutableHashTrieMap': 'System.Collections.Immutable.ImmutableDictionary',
'net.sf.saxon.ma.trie.ImmutableMap':        'System.Collections.Immutable.ImmutableDictionary',
'net.sf.saxon.ma.trie.ImmutableList':       'System.Collections.Immutable.ImmutableList',
'net.sf.saxon.ma.trie.TrieKVP':          'System.Collections.Generic.KeyValuePair',
...
'net.sf.saxon.s9api.Message':          'Saxon.Api.Message',
'net.sf.saxon.s9api.QName':
```

```

        'Saxon.Api.QName',
    'net.sf.saxon.s9api.SequenceType':
        'Saxon.Api.XdmSequenceType',
    ...
}"/>

```

Note that there are cases where we replace system classes with Saxon-supplied classes, and there are also cases where we do the reverse: the extract above illustrates that we can replace Saxon's immutable map implementation with the standard immutable map in .NET. In the case of `BigDecimal`, we rewrite the code to use a third-party library⁸ with similar functionality to the built-in Java class.

The other part of the rewrite process is to handle method calls. We rely here on knowing the target class of the method, and we typically handle the rewrite with a template rule like this (long namespace names abbreviated for space reasons: S.N = `Singulink.Numerics`, S.I.H = `Saxon.Impl.Helpers`)

```

<xsl:template match="*[@RESOLVED_TYPE = 'java.math.BigDecimal']"
    priority="20" mode="methods">
    <xsl:sequence select="f:applyFormat(., map{
        'add#1':      '(%scope%+%args%)',
        'subtract#1': '(%scope%-%args%)',
        'multiply#1': '(%scope%*%args%)',
        'divide#1':   'S.N.BigDecimal.Divide(%scope%, %args%, 18)',
        'divide#2':   'S.N.BigDecimal.Divide(%scope%, %args%)',
        'divide#3':   'S.N.BigDecimal.Divide(%scope%, %args%)',
        'negate#0':   '-%scope%',
        'mod#1':       'S.I.H.BigDecimalUtils.Mod(%scope%, %args%)',
        'signum#0':   '%scope%.Sign',
        'remainder#1': 'S.I.H.BigDecimalUtils.Remainder(%scope%, %args%)',
        'divideToIntegralValue#1':
            'S.I.H.BigDecimalUtils.Idiv(%scope%, %args%)',
        'divideAndRemainder#1':
            'S.I.H.BigDecimalUtils.DivideAndRemainder(%scope%, %args%)',
        'valueOf#1':   'Saxon.Impl.Helpers.BigDecimalUtils.ValueOf(%args%)',
        'intValue#0':   '((int)%scope%)',
        'longValue#0':  '((long)%scope%)',
        'doubleValue#0':
            '((double)%scope%)',
        'floatValue#0': '((float)%scope%)',
        'longValueExact#0':
            'S.I.H.BigIntegerUtils.LongValueExact(%scope%)',
        'setScale#1':   '%scope%', (:no-op, values are normalized:)
        'setScale#2':   '%scope%', (:no-op, values are normalized:)
        'stripTrailingZeros#0':
            '%scope%', (:no-op, values are normalized:)
        'toBigInteger#0':
            '((System.Numerics.BigInteger)%scope%)',
        '*':
            '%scope%.%Name%(%args%)'
    })"/>

</xsl:template>

```

This is a template rule in mode `methods`, a mode that is only used to process `MethodCall` expressions, so we don't need to repeat this in the match pattern. This particular rule handles all calls where the target class is `java.math.BigDecimal`. It delegates the processing to a function `f:applyFormat()` which is given as input

⁸<https://github.com/Singulink/Singulink.Numerics.BigDecimal>

a set of sub-rules supplied as a map in a custom microsyntax. Given the name and arity of the method call, this function looks up the applicable sub-rule, and interprets it: for example `value1.add(value2)` translates to `(value1+value2)` (C# allows user-defined overloading of operators such as "+"). Some methods such as `mod()` are converted into calls on a static helper method (written in C#) in class `Saxon.Impl.Helpers.BigDecimalUtil`.

Most of the product's dependencies have proved easy to tackle using one or more of these mechanisms. We were able to use rewriting more often than I expected – for example it's used to replace the dependency on Java's `BigDecimal` class with a third-party library, `Singulink.Numerics.BigDecimal`. It's worth showing the XSLT code that drives this:

5.2. Iterators and Iterables

There is a close correspondence between the Java interface `Iterable` and C#'s `IEnumerable`; and similarly between Java's `Iterator` and C# `IEnumerator`. In both cases the interface is closely tied up with the ability to write a "for each" loop. If we're going to be able to translate this Java:

```
for (Attribute att : attributes) {...}
```

into this C#:

```
foreach(Attribute att in attributes) {...}
```

then the variable `attributes`, which is an `Iterable` in Java, had better become an `IEnumerable` in C#. We can handle that by rewriting class names; and we can also rewrite the method `attributes.iterator()` as `attributes.GetEnumerator()` so that it satisfies the C# interface. What now gets tricky is that Java's `Iterator` has two methods `hasNext()` and `next()` which don't correspond neatly to C# `IEnumerator`, which has `MoveNext()` and `Current`. Specifically, `hasNext()` is stateless, and can be called any number of times, while `MoveNext()` is state-changing and can only be called once. However, "sane" code that uses an iterator always makes one call on `hasNext()` followed by one call on `next()`, and that sequence translates directly to a call on `MoveNext()` followed by a call on `Current`. So the converter assumes that the code will follow this discipline – and if we find code that doesn't, then we have to change it⁹.

5.3. Inner classes

Java effectively has three kinds of inner class: named static inner classes, named instance-level inner classes, and anonymous classes. Only the first of these has a direct equivalent in C#.

Saxon makes extensive use of all three kinds of inner class. The converter makes a strenuous effort to convert all of them to static named inner classes, but this doesn't always succeed. In some cases it can't succeed, because there are things that static named inner classes aren't allowed to do.

Sometimes the conversion can be made to work with the help of hints supplied as Java annotations. For example we might see the following annotation on a method that instantiates an anonymous inner class:

⁹A benefit of having the parsed Java code in XML format is that it's easy to do queries to search for code that needs to be inspected.

```
@CSharpInnerClass(outer=false,
    extra={"net.sf.saxon.expr.XPathContext context",
           "net.sf.saxon.om.Function function"})
```

This indicates to the converter that in the generated static inner class, there is no need to pass a reference to the outer `this` class (because it's not used), but there is a need to pass the values of variables `context` and `function` from the outer class to the inner class. (Annotations, like anything else in the Java source code, are parsed by `JavaParser` and made visible in the syntax tree.)

5.4. Overriding, Covariance, Contravariance

As we've already mentioned, C# requires any method that is overridden to be declared `virtual`, and any method that overrides another to be declared with the modifier `override`. We handle this by analyzing the class hierarchy and recording the analysis in the digest XML file, which is available to the stylesheet that generates the C# code.

In addition, Java allows an overriding method to have a covariant return type: if `Expression.evaluate()` returns `Sequence`, then `Arithmetic.evaluate()` can return `AtomicValue`, given that `AtomicValue` is a subclass of `Sequence`. C# doesn't allow covariant return types until version 9.0 of the language, and we decided this was a new promised feature that we would be unwise to rely on. Instead:

- ◊ when we're analyzing the class hierarchy, we detect any use of covariance, and change the overriding method to use the same return type as its base method;
- ◊ when we're analyzing the class hierarchy, we detect any use of covariance, and change the overriding method to use the same return type as its base method when we find a call to a method that's been overridden with a covariant return type, we insert a cast so the expected type remains as it was.

Java allows interfaces to define default implementations for methods; C# does not. The transpiler handles default method implementations by copying them into each subclass. This of course can lead to a lot of code duplication, so we have eliminated some of the cases where we were using default methods unnecessarily.

5.5. Generics

I've already mentioned that we identified early on that generics would be a problem, and one of the steps we took was to reduce unnecessary and unproductive use of generic types. In fact, we have almost totally eliminated all use of generics in Saxon-defined classes, which was the major problem. That leaves generics in system-defined classes (notably the collection classes such as `List<T>`) which we can't easily manage without.

In fact, most uses of these classes translate from Java to C# without trouble. But there are still a few difficulties:

Diamond Operators

Java allows you to write `List<X> list = new ArrayList<>()` (referred to as a diamond operator, though it's not technically an operator). In C# it has to be `new ArrayList<X>()`. So we need to work out what `X` is – essentially by applying the same type inferencing rules that the Java compiler applies. The way we do this is by recognising common cases: object instantiation on the right-hand side of an assignment, in a return clause, in an argument to a non-polymorphic method, etc. The logic is quite complex, and it catches perhaps 95% of cases. The remainder are handled by changing the Java code: either by introducing a variable, or by adding the type redundantly within the diamond.

XSLT template rules really come into their own here. We handle about a dozen patterns where the type of the parameter can be inferred, and each of these is represented by a template rule. As we get smarter or discover more cases, we can simply add more template rules. Here's an example of one of the rules:

```
<xsl:template match="*[@nodeType='ReturnStmt']
    [ancestor::member[1]/type/@RESOLVED_TYPE]/*">
    <xsl:variable name="type"
        select="ancestor::member[1]/type/@RESOLVED_TYPE"
        as="xs:string"/>
    <xsl:value-of select="f:extract-type-arguments($type)"/>
</xsl:template>
```

This rule detects a diamond operator appearing in a return statement (the rule appears in a module with default mode diamond, which is only used to process expressions that have already been recognised as containing a diamond operator). It finds the ancestor method declaration (`ancestor::member[1]`), determines the declared type of the method result, and inserts that into the C# code as the type parameter in place of the diamond operator.

Wildcards

The Java wildcard constructs `<? extends T>` and `<? super T>` have no direct equivalent in C#. The way we handle these depends on where they are used. The default action of the converter is just to replace them with `<T>`, which often works. But in class and method declarations we generate a C# where clause to constrain the type bounds, so

```
public class GroundedValueAsIterable<T extends Item>
    implements Iterable<T> {...}
```

becomes

```
public class GroundedValueAsIterable<T> : IEnumerable<T>
    where T : Item {...}
```

One issue we face is that the default type `Object` in Java is less all-embracing than the object type in C#: the former does not include primitive types such as `int` or `double`, the latter does. This means that where the required type is `Object`, the supplied value can be null; but this is not so in C#, because primitive types do not allow a null. This permeates the design of collection classes. Often the solution is to constrain the C# class to handle reference types only, using the clause `where T : class`.

5.6. Lambda Expressions and Delegates

Lambda expressions in Java translate quite easily to lambda expressions in C#: apart from the use of a different arrow symbol, the rules are very similar.

I've already mentioned that the `JavaParser` symbol solver struggles a bit with type inference inside lambda expressions, and we sometimes need to provide a bit of assistance by declaring types explicitly.

The main problem, however, is that Java is much more flexible than C# about where lambda expressions are allowed to appear. To take an example, we have a method `NodeInfo.iterateAxis(Axis, NodeTest)`. On the Java side, `NodeTest` is a functional interface, which means the caller can either supply a lambda expression such as `node -> node.getURI() == null`, or they can supply an instance of a class that implements the `NodeTest` interface, for example `new LocalNameTest("foo")`.

In C# `NodeTest` must either be defined as a delegate, in which case the caller must supply a lambda expression and not an implementing class, or it can be defined as a regular interface, in which case they can supply an implementing class but not a lambda expression.

To solve this, in most cases we've kept it as an interface, but supplied an implementation of the interface that accepts a lambda expression. So if you want to use a lambda expression here, you have to write `NodeTestLambda.of(node -> node.getURI() == null)`. Which is convoluted, but works.

5.7. Exceptions

The most obvious difference here between Java and C# is that C# does not have checked exceptions. Most of the time, all this means is that we can drop the `throws` clause from method declarations.

Try/catch clauses generally translate without trouble. A `try` clause that declares resources needs a little more care but we hardly use these in Saxon. The syntax for a catch that lists multiple exceptions is a little different, but the conversion rule is straightforward enough.

The main problem is deciding on the hierarchy of exception classes. If the Java code tries to catch `NumberFormatException`, how should we convert it? What exception will the C# code be throwing in the same situation?

To be honest, I think we probably need further work in this area. Although we're passing 95% of test cases already, I think we'll find that quite a few of the remaining 5% are negative tests where correct catching of exceptions plays a role, and we'll need to give this more careful attention.

6. XSLT Considerations

In this section I'll try to draw out some observations about the XSLT implementation.

Like most XSLT code, it has been developed incrementally: rules are added as the need for them is discovered. This is one of the strengths of XSLT as an implementation language for this kind of task: the program can grow very organically, with little need for structural refactoring. At the same time, uncontrolled growth can easily result in a lack of structure. How many modes should there be, and how do we decide? How should the code be split into modules? How should template rule priorities be allocated?

Again, like most XSLT applications, it's not just template rules: there are also quite a few functions. And as in other programming languages, the set of functions you end up with, and their internal complexity and external API, can grow rather arbitrarily.

It's worth looking a little bit at the nature of the XML we're dealing with. Here's a sample:

```
<member nodeType="MethodDeclaration">
  <body nodeType="BlockStmt">
    <statements>
      <statement nodeType="ReturnStmt">
        <expression nodeType="BinaryExpr"
          operator="PLUS">
          <left nodeType="MethodCallExpr"
            RETURN="double"
            RESOLVED_TYPE="net.sf.saxon.expr.Expression">
            <name nodeType="SimpleName" identifier="getCost"/>
```

```

<scope nodeType="MethodCallExpr"
       RETURN="net.sf.saxon.expr.Expression"
       DECLARING_TYPE="net.sf.saxon.expr.BinaryExpression">
    <name nodeType="SimpleName" identifier="getLhsExpression"/>
</scope>
</left>
<right nodeType="BinaryExpr" operator="DIVIDE">
    <left nodeType="MethodCallExpr" RETURN="double"
          RESOLVED_TYPE="net.sf.saxon.expr.Expression">
        <name nodeType="SimpleName" identifier="getCost"/>
        <scope nodeType="MethodCallExpr"
               RETURN="net.sf.saxon.expr.Expression"
               DECLARING_TYPE="net.sf.saxon.expr.BinaryExpression">
            <name nodeType="SimpleName"
                  identifier="getRhsExpression"/>
        </scope>
    </left>
    <right nodeType="IntegerLiteralExpr"
           value="2"/>
    </right>
</expression>
</statement>
</statements>
</body>
<type nodeType="PrimitiveType"
      type="DOUBLE"
      RESOLVED_TYPE="double"/>
<modifiers>
    <modifier nodeType="Modifier"
              keyword="PUBLIC"/>
</modifiers>
<annotations>
    <annotation nodeType="MarkerAnnotationExpr">
        <name nodeType="Name" identifier="Override"/>
    </annotation>
</annotations>
</member>

```

This represents the Java code

```

@Override
public double getCost() {
    return getLhsExpression().getCost()
        + getRhsExpression().getCost() / 2;
}

```

It's interesting to look at the values used (a) for the element name (e.g. body, left, right, expression, statement), and (b) for the nodeType attribute (e.g. ReturnStmt, BinaryExpr, SimpleName). Generally, the nodeType attribute says what kind of thing the element represents, and the element name indicates what role it plays relative to the parent. (Reminiscent of SGML architectural forms, perhaps?)

As an aside, the same dichotomy is present in the design of Saxon's SEF file, which represents a compiled stylesheet, but there we do it the other way around: if an integer literal is used as the right hand side of an addition, the JavaParser format expresses this as `<right nodeType="IntegerLiteral">`, whereas the SEF format expresses it

as `<IntegerLiteral role="right">`. Of course, neither design is intrinsically better (though the SEF choice works better with XSD validation, since XSD likes the content model of an element to depend only on the element name, not the value of one of its attributes). But the choice does mean that most of our template rules in the transpiler are matching on the `nodeType` attribute, not on the element name, and this perhaps makes the rules a bit more complicated.

Performance hasn't been a concern. I'm pleased to be able to report that of the various phases of processing, the phases written in XSLT are an order of magnitude faster than the phase written in Java; which means that there's no point worrying about speeding the XSLT up. This is despite the fact that (as the above example demonstrates) the XML representation of the code is about 10 times the size of the Java representation.

(Actually, the Java code is 29Mb, the XML is 120Mb, and the generated C# is 18Mb. The C# is smaller than the Java mainly because we drop all comments, and also because the Java total includes modules we don't (yet) convert, for example a lot of code dealing with SAX parsers, localisation, and optional extras such as the XQJ API and SQL extension functions).

But I would like to think that one reason performance hasn't been a concern is that the code was sensibly written. We've got about 200 template rules here, most of them with quite complicated match patterns, and we wouldn't want to be evaluating every match pattern for every element that's processed. In fact, a lot of the time we're doing three levels of matching:

- ◊ If we find that we're processing a method call (which is rather common), we have a single template rule in the top-level mode that matches `*[@nodeType='MethodCallExpr']`.
- ◊ This template rule then does `<xsl:apply-templates select=". " mode="MethodCall"/>`, which searches for a more specific template rule, but only needs to search the set of rules for handling method calls, because they are all in this mode.

To make the code manageable and maintainable, we put all the template rules for a mode in the same module, and use the XSLT 3.0 construct `default-mode="M"` to reduce the risk of accidentally omitting a mode attribute on a template rule or `xsl:apply-templates` instruction.

- ◊ Most of the template rules for method calls are structured as one rule per target class; as described earlier, this uses a microsyntax for defining the formatting of each possible method, using XSLT maps.

So it's not a flat set of hundreds of rules; we've used modes (and the microsyntax) to create a hierachic decision tree. This both improves performance, and keeps the rules simpler and more manageable. It also makes debugging considerably easier: as with any XSLT stylesheet, working out which rules are firing to handle each input element can be difficult, but the splitting of rules into modes certainly helps.

(A little known Saxon trick here is the `saxon:trace` attribute on `xsl:mode`, which allows tracing of template rule selection on a per-mode basis).

7. Conclusions

Firstly, we've confirmed the viability of using XSLT for transformation of abstract syntax trees derived from parsing of complex grammars, on quite a significant scale. The nature of XSLT as a rule-based pattern matching language makes it ideally suited for such tasks. It's hard to imagine how the pattern matching code would look if it were written in a language such as Java; it would certainly be harder to maintain¹⁰.

At the same time (and perhaps not quite so relevant to this particular audience, but significant nonetheless) we've demonstrated a pragmatic approach to transpilation. Without writing a tool that can automatically perform 100% conversion of any Java program, we've written a set of rules that works well on the subset of the Java language (and class library) that we're actually using, and more importantly, done so in a way that allows manual intervention to handle the parts that can't (cost-effectively) be automated, without sacrificing repeatability - that is, the ability to re-run the conversion every time the Java code changes. And by writing the rules in XSLT, we've created a transpiler that is readily capable of extension to cover features that we chose to leave out first time around.

I take satisfaction in the quality of the generated C# code. It's human readable, and it appears to be efficient. This is achieved partly by the policy of not worrying too much about edge cases. By doing our own customised conversion rather than writing a product that has to handle anyone's Java code, we can be pragmatic about exactly how faithfully the C# code needs to be equivalent to the original Java in edge cases.

¹⁰While writing the paper, I discovered (without surprise) that transpilation using XSLT has been done before: see <https://www.ijcrt.org/papers/IJCRT2005043.pdf>. That paper, however, gives little detail of how the conversion is done, and appears only to tackle trivial code fragments.



Comprehensible XML

Markup UK 2021

Erik Siegel, Xatapult

Writing software, it is all too easy to forget that there is another side to it than just: it works. Most software goes through a life cycle of writing, testing debugging and maintenance. This makes it important that what you write is comprehensible, both for somebody else and yourself in a few months' time. It reduces the chance of mistakes and bugs and shortens development time.

We have probably all seen, heard, or read something about how to write good code. We probably all try to comply more or less, but given what we see around us, we do not always succeed.

This talk will try to provide directions, tips and tricks on how to make (in this case XML related) code more understandable. It will also provide background on why this is important and why we should try to comply. How can we do this with minimum effort. It will be a mixture of things from literature and personal experience after 40 years of programming computers.

1. Introduction

Writing software, it's all too easy to forget that there is another side to it than just "it works" or "it's fast". Most software, only throw away scripts are probably exempt, goes through a life cycle of writing, testing debugging and maintenance.

This makes it important that what you write is comprehensible, both for somebody else and yourself in a few months' time (or less). It is not just a matter of being nice: intelligible code, being able to easily grasp what is meant, reduces the chance of mistakes and bugs, and shortens development time.

Nobody is in the business of deliberately writing puzzling code. But given what we probably all encounter now and then, there is a lot of jigsaw software in the world. What is this XSLT trying to do? Why the %\$#^ is this XQuery not properly indented, so I can see what else belongs to what if? What does this variable store and what's its type? What am I supposed to pass to this parameter?

We all forget to pay attention to these kinds of things. We were hurried, hungry, caffeine-depleted, tired, or stressed. We thought this piece of software would be thrown away, but it miraculously survived for many, many years. We were in a flow and wanted to see an end-result quickly. Even although we knew that a little attention now would save many, many hours of maintenance and bug-hunting later, we did not pay enough attention.

So, what we can do about it? We have probably all seen, heard, or read something about how to write good code. We probably all try to comply, more or less, but given what we

see around us, we do not always succeed. We are *not* the computers we program; our head contains a different kind of CPU and we must deal with that.

This talk will try to provide directions, tips and tricks on how to make code more understandable. It will also provide some background on why this important and why we should try to comply, despite all the excuses we give ourselves every day. How can we do this with minimum effort. It's a mixture of things from literature and personal experience after 40 years of programming computers.

Comprehensible code is not something specifically for XML alone, so the applicability is therefore wider. Examples will be for XML programming languages.

1.1. About the author

I'm an XML specialist, doing things like consulting, designing and programming, strictly XML technology (like XSLT, XQuery, Schemas, Schematron, etc.). Before that I worked as a programmer and system architect, using languages like assembler, C, Visual Basic, PERL and several others.

Besides the technical side of things I also like communicating about it. I'm the author of two books: one about eXist-db (together with Adam Retter) and one about XProc 3.0. There are several articles I have written on <https://www.xml.com>.

More about me at <http://www.xatapult.com>.

2. Why bother?

Unless you're a super-human programmer, you've written software that contains bugs. Small and obvious ones that were found during your own test runs or more hidden ones that suddenly raised their ugly heads in production. There are entire books and conferences devoted on to how to avoid this.

Besides things in the large (design, architecture, module structure, etc.), there are also a lot of things in the small that can help to avoid or more easily detect problems. Making your code more *comprehensible* is one of them. It'll help yourself creating solid code and will be a sight for sore eyes for the maintenance people (yourself?), later on in the product's life-cycle.

You can shrug about this, but bugs can be very costly. Or worse: deadly. If you wanna have "fun", read the Wikipedia page about bugs that matter: https://en.wikipedia.org/wiki/List_of_software_bugs. Or this one, that tries to identify the worst software bug in history: <https://www.laserfiche.com/ecmblog/whats-worst-software-bug-history/>. Or remember the very deadly problems with the Boeing 737 MAX, partly caused by a few lines of code.

Figure 1. The spectacular explosion of the Ariane 5 flight in 1996, caused by a software bug...



Of course we don't all write rocket, flight guidance, medical or otherwise critical stuff. What we do is usually very... non-critical, ordinary, boring, quotidian? Still, helping yourself and others to easily comprehend what a piece of software is supposed to do is not only just being nice. Some reasons for caring about this:

- ◊ Whether the software is critical or not, bugs are a nuisance. And understandable code helps detecting them. By others but also, or maybe even foremost, by yourself.
- ◊ It's a known fact that most software spends way more time being maintained than initially written. Much of this maintenance time is spent trying to *understand* what the software is trying to do, and how. Anything we can do to help with this is pure profit.
- ◊ Software is costly, a few hours spend on comprehensibility upfront saves many, many hours later.
- ◊ Professional pride...

With my focus here on things we can do during the actual writing of the code, I'm not saying that any measures in the large don't matter. They do, a lot. But since I spend most of my time in the (XML) software writing trenches, this is what I know most about. And since I encounter a lot of code where things are wrong (yes, also by myself), a little refresher can't hurt.

3. What can we do?

This section contains the measures that IMHO (author's prerogative) are most important in making code comprehensible. Most, if not all, is backed up by software engineering literature. However, what is dealt with here is subjective, both in choice of subjects, examples and proposed solutions. You might disagree or think I missed some key point. I hope nonetheless we don't disagree about the *importance* of making code comprehensible. We probably all struggle with this and try to do the best we can. Let this all be at least be a source of inspiration and awareness.

3.1. Convention yourself

If there's one thing you can do yourself a lot of fun with, it's *conventions*. Huh, conventions? Aren't that these long and boring lists of things you need to comply with, from variable names to comment structure? Which you never ever seem to be able to adhere to in full? Yes, that's what I mean!

There are two main reasons why conventions are important:

- ◊ They provide more "thinking space". What if you had to come up with a variable naming convention every time you created one? What if you had to decide how to format a function header on every occasion you started one? You don't want that. Following a pattern, aka convention, is way easier to your working memory, already filled with the many intricate details of a program's logic.
- ◊ It boosts comprehensibility. Being able to see what this variable name stands for, where a function starts and ends, what else belongs to what if, helps a lot in understanding a piece of software.

I admit, conventions have a bad name. Sometimes they're used to wrap long red tapes around programming efforts, which reduces all the fun. But applied properly they can absolutely help improving the comprehensibility of the code. How can we make them effective?

- ◊ There should't be too many of them. It must easily fit in your head. So maybe a list of one or two pages long?
- ◊ They should't be regarded as *absolute*. When following a convention leads to ridiculous long names/indenting/indecipherable code/overly long functions/... (cross out what does not apply), either change the convention or decide that this is an exceptional case and break the rules.
- ◊ And yes, you have to get used to them. If the list isn't too long and the conventions more or less make sense, they'll soon become a habit.

With respect to comprehensibility, what are the basic things important enough to have a convention about?

- ◊ Names of variables, functions, data structures, etc.: Pick something and stick to it. I personally prefer the `lower-case-with-hyphens` convention but there's nothing sacred about it. The `lowerMixedCase`, `UpperMixedCase`, `lower_case_with_underscores` or whatever are just as good. You can also use multiple naming styles for different things in your program, but don't make choosing the right one too complicated!

The only thing I would definitely *not* recommend is using something without some kind of word separation. The worst example probably being `ALLCAPSWITHOUTANYSEPARATORS`: too hard to read and understand.

- ◊ Layout: Things like indentation, whitespace, empty lines, separators, comment styling, etc. Choose something and try to keep the code consistent. More about this in Section 3.2 [53].

You can make it easy on yourself here: find out what the pretty-print functionality of your IDE is capable of, and use *that* as (part of) your layout convention...

- ◊ Commenting: What to write comments about, how they're formatted, etc. See Section 3.4 [57].
- ◊ Filenames and directory structures: Again, pick some naming convention. Don't forget to standardize the *extensions* (is an XQuery script `*.xq`, `*.xql` or `*.xquery`?). If there isn't already one, try to invent a convenient directory structure, using standardized names, that fits the bill.

3.2. The rhythm of the code

Layout is extremely important for the comprehensibility of the code. $2+3 * 1+2$ is *not* 15 (what the spacing suggests) but 7. Or what about this coding horror:

```
<xsl:function name="local:something">
<...></xsl:function><xsl:function name="local:something-else"><...>
</xsl:function>
```

What are the things we can do to enhance comprehensibility? First a few tips about expressions and the likes:

- ◊ Use parentheses abundantly. Write $(2+3)*(1+2)$ if that's what you mean. Parentheses help to communicate the intention of your expressions.
- ◊ Choose a style about where to add spaces. Do you write: $(2+3)*(1+2)$ or $(2 + 3) * (1 + 2)$? Format a function call like `add(1, 2)` or `add(1, 2)` or `add(1, 2)`... Again, pick something, stick to it.

- ◇ If there's any syntactic sugar in your language that makes expressions easier to read, by all means, use it. For instance, since version 3.0, XPath has the `=>` operator. So instead of:

```
lower-case(normalize-space(translate($name, '\', '/')))
```

We can now write the, IMHO much clearer:

```
$name => translate('\\', '/') => normalize-space() => lower-case()
```

But what is, in my eyes, most important is the code's "rhythm". When you look at some script or module, can you easily see how it's structured? Where things begin and end? What belongs to what? Here are my tips:

- ◆ Choose an indentation style and try to keep this consistent.

For XML based languages like XSLT that's easy: indenting of XML is already (unofficially) standardized. The only thing we can bicker about is *how many* to indent and whether to use spaces or tabs (I personally prefer an indent of two spaces and not to use tabs).

For text based languages like XQuery it's a little harder. I've done a lot of XQuery programming lately and noticed I couldn't come up with a consistent and simple enough indentation strategy that works in all situations. Especially for XQuery, make sure that it's clear what code belongs to what FLWOR expression or if/then/else branch. Whatever you choose, it's always making the *intention* of the code clear that matters most.

- ◇ Open some code module in your editor, scroll a bit, and look at it as from afar. Can you see how it's structured? What the building blocks are? When you're looking for some template/function/section in the code, can you quickly find it? And in a glance see where things start and end?

Clearly separate the main parts of your code. For instance, an XSLT program consists mainly of templates. Make it very clear where one begins and ends. Create a *sectioned layout*, like chapters/sections in an article or book.

I personally prefer using “comment lines” for this:

And use a “heavier” line in between sets of templates that belong together:

And no, it's *not* awkward to insert (type) these kinds of lines all the time, see Section 4 [59]. But you can also decide to use multiple empty lines. Or whatever works for you.

- ◇ Use empty lines and comment headers to give code inside a template/function/... a "rhythm":

```
(: Initialize: :)
...
(: Compute the value for ... :)
...
(: Write it to disk: :)
...
```

Splitting code in blocks like this serves two main purposes, analogue to paragraphs in prose:

- ◆ They force you to think more structured and do the things that belong together together.
- ◆ Somebody which is new to the code can more easily grasp what it's about (especially when the comment headers are helpful, see Section 3.4 [57]).

The code blocks should't be very long, max. 10 to 20 lines, preferably shorter. Just a single line is ok if it serves the purpose.

- ◇ Use a maximum line width to prevent your lines from overflowing/wrapping and making them hard to follow. Older books about software engineering advocate 80 characters. Given our modern big screens and the tendency not to print things, I personally prefer ~150.

Using a line width can even flag incomprehensibility issues! When a piece of code starts regularly overflowing this, it's usually a clear indicator the code is too complex and you'd better refactor/split it...

3.3. Names, names, names (and declarations)

Choosing the right name for something (variables, functions, templates, etc.) is extremely important. A function named `computeIt` doesn't really communicate much meaning. If we decide to call it `computeTaxForCustomer`, its intent is much clearer. Naming matters.

Here are some comprehensibility measures with regards to naming:

- ◇ Don't be shy of using long and descriptive names. We're a long way from computer languages that forbid names longer than 8 or 16 characters. Most IDEs support you by providing pop-up lists of names to choose from if you want to refer to a variable, function or template (and even if not we have copy/paste!).
- ◇ Use descriptive names. This is the absolute winner in making code self-documenting. For instance:

```
<xsl:variable name="f" as="xs:double" select="($c - 32) * 0.5556"/>
```

It takes a comment to tell the reader that this converts a temperature in Celsius to Fahrenheit. But if the variable names were chosen more wisely this is no longer necessary:

```
<xsl:variable name="temperature-in-fahrenheit" as="xs:double"
    select="($temperature-in-celsius - 32) * 0.5556"/>
```

Or even better: break out such an expression in a function:

```
<xsl:function name="mod:celsius-to-fahrenheit" as="xs:double">
  <xsl:param name="temperature-in-celsius" as="xs:double"/>
  <xsl:sequence select="($temperature-in-celsius - 32) * 0.5556"/>
</xsl:function>
```

- ◊ Break a long and complicated expression into smaller parts using aptly named variables.
For instance:

```
if (($status eq $status-success) or
    (($amount ge $amount-limit) and (not($special-account-type))))
then ...
```

We can make its intent more clear if we rewrite it to:

```
let $successful-attempt as xs:boolean := $status eq $status-success
let $large-withdrawal-permitted as xs:boolean :=
  ($amount ge $amount-limit) and (not($special-account-type))
return
  if ($successful-attempt or $large-withdrawal-permitted)
  then ...
```

Don't worry: only in very rare circumstances you have to be concerned about the performance impact creating a bunch of additional variables. And probably, the compiler will optimize them away for you.

- ◊ If something is in a certain unit (meters, inches, kilograms, dollars, zorkian foepies), stick the unit to the name. For an example see the Celsius to Fahrenheit conversion above.

I specifically mention this because of a bug classic: a software mismatch between the metric system and the English measurement system caused the NASA Mars Climate Orbiter to crash in 1998, at the cost of \$125 million (<https://www.simscale.com/blog/2017/12/nasa-mars-climate-orbiter-metric/>).

- ◊ Magic values, numeric and string constants with special meanings, should be given a name. Declaring such magic values centralizes their definition, which makes them easier to lookup and/or change and prevents bugs caused by mistyping a value. A good name clearly communicates the intent of the value.

Here's a classic example:

```
<xsl:for-each select="1 to 12">
```

Why 12? Ah, this is better:

```
<xsl:for-each select="1 to $months-per-year">
```

It's not that I expect that the number of months in a year is going to change anytime soon. This is about communicating the intent: we're iterating over months here...

Sometimes there is grumbling (me included) about having to create long and boring lists of magic name declarations, like this:

```
declare variable $mod:status-error as xs:string := 'error';
declare variable $mod:status-warning as xs:string := 'warning';
...
```

That looks superfluous, until you make a hard to spot "typo" bug:

```
if ($status eq 'error') then ...
```

- ◊ A convention for building the names is also good. Things like:
 - ◆ Which abbreviations can be used (max, min, ptr, etc.)
 - ◆ Use something like object-action (file-read, status-get, etc.) or action-object (read-file, get-status, etc.)
 - ◆ Special prefixes for booleans (is-..., do-...).
- ◊ If you declare something, make that declaration as *complete as you possibly can*. For a variable or parameter, *always* specify its type. For a function, always specify the return value's type. Even XSLT named templates can declare a return type (in an as attribute); specifying this is unusual but definitely not wrong!

Always specifying datatypes has double benefits. It tells you more about the declaration, increasing its comprehensibility. It will also cause a whole bunch of errors to surface sooner if you make mistakes.

3.4. Comments? What comments?

Probably everyone knows the famous saying “better no comment than a wrong one”, or one of its variations. And it’s true: it’s extremely confusing reading what a piece of software is supposed do and find out it does something different... Of course, it happens to all of us. You wrote something once, including informative comments. Then you refactor it, then refactor some more, and in having to deal with all the complexities of getting it to work, forget to update the comments.

Here are some best practices for commenting:

- ◊ Make a clear distinction between *black-box* and *white-box* comments.
 - ◆ Black-box comments are things that should be understandable without having to consult the code. For instance module, template and function descriptions. These comments must still make sense when they’re separated from the code, as for instance happens with XQuery xqDoc (<http://xqdoc.org>) comments (between (:~ ... :)) in some environments.
 - ◆ White-box comments are *about* the code. These comments should help the reader to grasp what’s going on.
- ◊ Do not comment the obvious, comment the *intent*. This, for instance, is completely superfluous:

```
<!-- Store the number of <thing> elements in $things-count: -->
<xsl:variable name="things-count" as="xs:integer" select="count(/things)"/>
```

Both the (well-chosen) variable name as the expression already tell you this. This however tells the reader what the intent of the code is:

```
<!-- Add a prompt attribute to every thing for easier reporting later: -->
<xsl:variable name="things-count" as="xs:integer" select="count(/things)"/>
<xsl:for-each select="/things">
  <xsl:copy>
    <xsl:attribute name="prompt" select="position() || '/' || $things-count"/>
    <xsl:apply-templates select="@* | node()" />
  </xsl:copy>
</xsl:for-each>
```

```
</xsl:copy>
</xsl:for-each>
```

- ◊ Use comments (and empty lines) for creating “rhythm”. See Section 3.2 [53].
- ◊ Use comments as the equivalent of *section titles*, even if what the comment says is obvious when you *know* the code. Remember, the reader isn't always that knowledgeable.

```
<!-- Initialize: -->
...
<!-- Compute the values for ... -->
...
<!-- Done, wrap up: -->
...
```

This creates an additional, smaller, rhythm inside the bigger rhythm of templates and functions.

- ◊ Some movies on DVD (when we used to have DVDs, old-man's reminiscences) had the option to turn “director's comments” on. You then heard the movie director in a voice-over about certain scenes: how they were taken, why certain choices were made, etc. One of the best pieces of advise about commenting I ever had was that comments should be exactly like that: like a director telling you about the code.
 - ◆ Why is this code different from the rest? (bug workaround...)
 - ◆ Why this completely incomprehensible expression? (performance...)
 - ◆ What you did to make this working? (prevent problems already solved...)
 - ◆ What trap not to fall into when maintaining this code? (point out obvious mistakes...)

Fantasizing about being a famous movie director and providing comments about your brilliant artistic choices is a constructive mindset for creating high quality comments.

- ◊ Be very careful with blocks of commented-out code. Don't make the reader guess about why you left it in. Forgotten to delete? Laziness? Carelessness? Is this still important? Can I delete it now or was there a good reason to keep it? At least provide some comment about the why if you really think it still serves a purpose.

3.5. It's the process...

In a presentation for MarkupUK 2019 called *Documenting XML structures*, I introduced the term “knowledge bubble” (<https://markupuk.org/2019/webhelp/index.html#ar12.html>). What I meant is that when you're busy doing something complicated, like programming, it's very hard to imagine what people on the outside don't know or will understand. You're in the middle of it now and everything is therefore understood and crystal clear.

You have to acknowledge to yourself that you're *in a knowledge bubble* during development. And being in a knowledge bubble means it's impossible to write good documentation because you don't really realize what the reader doesn't know. This is not only true for documenting things (in separate documentation), but also for writing good code comments *and* judging code comprehensibility.

So how can we overcome this obstacle? Here's how I try do this:

- ◇ I write my software with in the back of my mind that I will come back to it later. That makes it OK (and not laziness) to flag missing comments and sometimes missing non-essential pieces of code with a *TBD* marker (or whatever you want to use meaning *To Be Done*).

Adding these TBD markers is important so you won't forget or overlook anything later. Writing a function but not feeling like adding the descriptive header comment? Add a comment with TBD. Having to write some error handling code that is not essential during initial development? Add a comment saying TBD. I think you get the picture.

- ◇ Find some time, later that week, maybe after a weekend, to review your code and fill in the TBDs. Try to do it when the knowledge bubble has deflated to a reasonable size (so not *immediately* after finishing coding).

Even if the code is flawless (which never happens), you'll be astonished what you find with regard to comprehensibility issues.

Key is that after you *think* you're done programming, take a pause/break/weekend/vacation/cup of tea and *always* revise/review what you've written. And also: important commenting is best left to review time.

4. Debunking some obstacles

When I talk with people about this, I hear two counterarguments over and over. Let me try to debunk them, once and for all.

- ◇ "This costs too much performance..." Nonsense:

- ◆ On our modern hyper-fast machines you won't notice those few milliseconds for creating an extra variable or performing that additional function call. If it takes time anyway, because...
- ◆ Compilers/interpreters are smarter than you imagine and optimize a lot of your troubles away.
- ◆ There will always be exceptions, usually loops that iterate so often you have to squeeze out the very last millisecond. If that's the case, of course, just do it. But please comment any weird, unusual or indecipherable construction lavishly, so it becomes comprehensible again.

- ◇ "XML comments, lines and stuff like that are too many keystrokes, it slows me down...". Nonsense:

In most IDEs you can put stuff like starting a comment, inserting a line or whatever under some ctrl/alt/shift/apple combination. For instance, on my system inserting a comment is just a single keystroke away and leaves the cursor in between the <! -- - -> markers, ready to type!

It's a matter of finding out how this works in your environment and spend an hour or so customizing things. Well worth the effort.

5. Further reading?

There are numerous books about software engineering in general but only a few about coding itself. The ones I know of:

- ◇ Code Complete; Steve McConnell; Microsoft Press; 2004.

◊ Clean Code; Robert Martin; Addison-Wesley; 2009

Both still available and warmly recommended.



How Much Tag Set Documentation is Needed?

How much is too much?

Debbie Lapeyre, Mulberry Technologies, Inc.

The more documentation the better. Documentation is expensive, stick to the basics. If it isn't well documented people won't use it or, worse, won't use it consistently. We can't afford better documentation. We've all heard the cliches.

JATS (The Journal Article Tag Suite) has documentation — A LOT of documentation. Documentation designed to introduce new users to the tag set. Documentation designed to support experienced users. Documentation to support people who are customizing JATS, including advice on both the mechanics and the logic of making customizations. There are definitions, helpful remarks, tagged examples, extended essays. There is an International Standard that meets political needs and a site with non-normative documentation that meets practical needs. There are third party sites advising users on how to use the tag sets for best interoperability, and many organizations that ingest JATS provide (and may insist on) site-specific local rules.

It is entirely possible that JATS is the most heavily documented XML tag set of all time. Do other tag sets need this much documentation? Are there techniques other tag sets would find useful? After a guided tour of the JATS documentation, the audience can chime in: How much documentation does a tag set need? Do any tag sets need this much documentation? How much documentation does YOUR tag set need? What is useful? IS any of this overkill? What audience most needs to be served? What parts of the Tag Library you like? What parts not so much? How can we improve the JATS documentation?

1. Introduction

1.1. What is an XML “Tag Set”?

I define “XML Tag Set” as an XML vocabulary or a domain-specific set of elements; such a tag set is a named collection of XML elements and attributes that describe at least one document type.

Some popular examples include:

- ◊ DocBook
- ◊ JATS (Journal Article Tag Suite)
- ◊ TEI (Text Encoding Initiative)
- ◊ STS (Standards Tag Suite)
- ◊ BITS (Book Interchange Tag Suite)

- ◊ DITA (Darwin Information Typing Architecture, which includes a tag set and a lot more)
- ◊ Akoma Ntoso (an expandable tagset)
- ◊ NIEM (National Information Exchange Model)

Notice that I have defined this as a “vocabulary” not as a “language”. Vocabularies are descriptive rather than operational. XML vocabularies (tag sets) identify document components and data structurally and semantically. By this definition HTML is a tag set, but XSLT (which is also a set of tags) is not.

Further, when I talk about documentation, I mean textual and graphic material for reading by human agents. Machines, programming languages, and artificial intelligent agents need very different documentation.

1.2. The Documentation Cliches

Nothing has been as written about and over-cliched as documentation. We've all heard the positive cliches:

- ◊ More documentation the better.
- ◊ If a tag set isn't well documented, people won't use it.
- ◊ If a tag set isn't documented, people will not use it consistently.
- ◊ Interchange requires documentation.
- ◊ Databases require documentation of ingest format.

There are negative documentation cliches too:

- ◊ XML is self-describing. (HA!)
- ◊ Documentation is too expensive.
- ◊ We can't afford *better* documentation.
- ◊ We are agile, moving too fast to document.
- ◊ It's just a frill, stick to the basics.
- ◊ Takes too long; we'll make it better in Phase II.
- ◊ We don't need no stinking documentation!

There are also cliches that I believe in, because documentation is, in my opinion critically important.

- ◊ Tagging systems need consistency, for both downstream processing and interchange.
- ◊ We are told that XML will live a long time, and so it will, and usefully, if we understand what it means.
- ◊ If there is not fundamental agreement on how something should be tagged, we are sunk.

This is *why* we document tag sets — for people, for instruction in tagging, for tagging consistency.

2. Planning Considerations for documenting tag sets

Many considerations/vectors/parameters must be taken into consideration in documenting a tag set. Among the most important are the typical who, what, and why:

- ◊ Scope of the tag set (what documents it covers);
- ◊ Scope of the documentation (may be far less);
- ◊ Purpose of the documentation; and the
- ◊ Management, governance and support;
- ◊ Audience for the documentation.

2.1. Scope of the documentation

Exactly how much of a tag set are you documenting? Minimally, this means deciding:

- ◊ What document type(s) does this documentation cover? (For example, JATS provides a vocabulary for describing the textual and graphical content as well as the bibliographic metadata of journal articles, including research articles; subject-review articles; non-research articles; letters; editorials; book, software, and product reviews; peer reviews, and author responses included with an article. The Tag Set allows for descriptions of the full article content or just the article header metadata.)
- ◊ What document type(s) does this documentation not cover (by design)? JATS does not provide the elements to describe an issue of a journal, a book, a report, or a standard.
- ◊ Do you define included vocabularies? Is that tag set using namespace metadata from TEI or Dublin core? Does the tag set include MathML? Do you write explanations for these external vocabularies? JATS tag sets may include MathML, XHTML tables, OASIS tables, SVG graphics, etc. JATS does no document MathML, CML (Chemical Markup Language), or SVG. JATS Tag Libraries do provide element and attribute documentation for the XHTML table model, and a separate library for the OASIS CALS Table model.
- ◊ Do you define all of your tag set or just a portion? Particularly if the tag set you are documenting is drawn from a larger set of elements, it is vital to draw your boundaries.

2.2. Purpose

Purpose encompasses both the purpose of your tag set and the specific uses envisioned for your documentation.

Purpose of tagset — For example, the purpose of the JATS Publishing Tag Set (Blue) is “to provide a standardized format for users to regularize and optimize journal article data for exchange with publishing, hosting, or archiving partners. The JATS model works as a conversion target for article content provided both in XML and non-XML formats.” The purpose of the JATS Archiving and Interchange Tag Set (Green) is “to provide a standardized format in which to preserve the intellectual content of previously-published journal articles, capture structural and semantic components, and provide a single format into which content from many providers can be translated easily, with minimal loss.”

Both of these are descriptive not a prescriptive missions, therefore, JATS is an enabling tag set, it does not enforce!

Purpose of documentation — Documentation also has a purpose, which should be decided *before* work begins. The primary purpose of tag set documentation might be:

- ◊ Reference (like a programming manpage)
- ◊ Tutorial (starting, getting your head around the tag set)
- ◊ Tagged samples to copy (no text at all)
- ◊ Conference introduction or sales pitch
- ◊ Omnibus (all things to all people)

The worst purpose is the last: one omnibus documentation set to rule them all. JATS Tag Libraries are mostly intended for reference, but contain some tutorial aspects.

2.3. Management, Governance and Support

For any tag set, it is critically important who makes the decisions and where the money comes from. Anybody cannot just modify a tag set at any time. There needs to be a process for requesting changes and a process of approving changes and Quality Assurance. Somebody has to pay (this stuff ain't free!) for both creating and maintaining a tag set. Maintenance costs can be very long term (JATS is nearly 20 years old). Documentation needs to be a line item (explicitly) in someone's budget.

There are also budget implications for who writes the documentation. Technical writers are cheaper than vocabulary developers (on the whole) but may need more management and guidance. Technical people such as programmers will know all the answers, but may not be as articulate as Tech writers. Using one group to rework the prose or check the other can be a useful strategy.

JATS employs a hybrid model, with volunteers at NISO controlling the technical content of JATS and with the non-normative supporting materials, (schemas, samples, and Tag Libraries) funded and hosted by the United States National Library of Medicine (NLM). In practice, JATS users request tag set changes on a comment form on the NISO site. The JATS Standing Committee, chaired by Jeff Beck of NLM and Tommie Usdin of Mulberry Technologies, meets regularly by Zoom to determine solutions to technical issues. The schemas (DTD, XSD, and RNG) and the Tag Library documentation are maintained by the JATS Secretariat (Mulberry Technologies, Inc.).

2.4. Audience

The nature of the intended audience can change dramatically the vocabulary, coverage, tone, and even the layout of documentation. Considerations include:

Level of audience experience — Documentation must walk a fine line between being easy enough for a complete novice or an irregular user (once or twice a year) to use, while not annoying your every-day-like-clockwork user. The important user categories are divided by amount of documentation use not by the amount of tag set experience or technical expertise of the user:

- ◊ first time user
- ◊ casual user
- ◊ frequent user

I do not mean that it is irrelevant what kind of technical experience the users have. XML, tag set, and programming experience will all determine the approach and vocabulary used in the documentation. The documentation users, who are they:

- ◊ Document people (editors, authors)
- ◊ XMLers (technical, but know documents)
- ◊ Script writers, developers, general programmers (who might be quite technical but not know documents or XML)

3. JATS Documentation

The Journal Article Tag Suite (ANSI/NISO standard [*ANSI/NISO JATS Version 1.2 (Z39.96-2019)*]) establishes elements and attributes from which tag sets for multiple document types can be built. There are three official tag sets in JATS family:

- ◊ JATS: THE standard for tagging journal articles (worldwide). JATS Article Tag Suite has been developed in three “flavors”: Archiving, Publishing, and Authoring.
- ◊ STS: Tag set for standards (Standards Tag Suite). STS has been developed in two “flavors”: Interchange and Extended.
- ◊ BITS: Tag set for books (Book interchange Tag Suite)

Each tag set has its own documentation, sharing software and editorial style, but not content.

JATS has two “official” sets of documentation:

- ◊ The ANSI/NISO standard [*ANSI/NISO JATS Version 1.2 (Z39.96-2019)*]
- ◊ The “non-normative” documentation, which consists of:
 - ◆ XSDs, DTDs, RNGs
 - ◆ Tag Libraries (documentation)
 - ◆ Sample tagged documents

It is very important that the ANSI NISO standard exists, since many organizations can only use tag sets that are national or international standards. That said, not many people read the text of the standard, as it contains only a subset of the information in the Tag Libraries, and is not as well linked or indexed. People read about and find references for JATS through the non-normative documentation. The ANSI/NISO standard and the Tag Libraries are produced from the same XML source, so the information they have in common really is common.

In addition to the official documentation (described in this paper), there is an even greater volume of user-built documentation, guidance, and Best Practice recommendations. Third party user groups and organizations that ingest or process JATS have also produced frameworks, stylecheckers, output transformations, and other automated tools. See the *Other Resources to Support JATS Users* Section for a very partial list.

3.1. Why talk JATS documentation?

To be honest, I am talking about JATS because it is the documentation I know best for the tag set I know well. But there are good reasons to examine what has been done with JATS. People have stated (in public at conferences) that the reason JATS has been so successful is its documentation. The documentation far exceeds that of the immediate predecessors to JATS; AAP DTD and ISO-12083.

My other rationale for this paper is that I would like feedback from this community, concerning how the JATS documentation can be improved. Most conference attendees

have worked with many tag sets, most of them non-JATS. What can these communities learn from each other?

4. Diving into the JATS Documentation

Separate Tag Libraries for JATS Archiving, Publishing, and Authoring (the three flavors of JATS) are available at [JATS Homepage] to use interactively or for download for local interactive use. JATS Tag Libraries are large numbers of linked HTML files. Unlike the old days of tag set documentation, nobody prints this material and nobody writes books interpreting this material. JATS reference happens online through browsers. [Note: If, however a JATS user chooses to print one of the HTML pages (all about one element or an essay), some very nice CSS turns the HTML the user is viewing into better-looking pages for print. All part of the service.]

JATS Tag Libraries are listed by name and color-coded to make it easy for users to know which Tag Library they are viewing.

Figure 1. Archiving and Interchange (Green) Tag Library



Figure 2. Journal Publishing (Blue) Tag Library

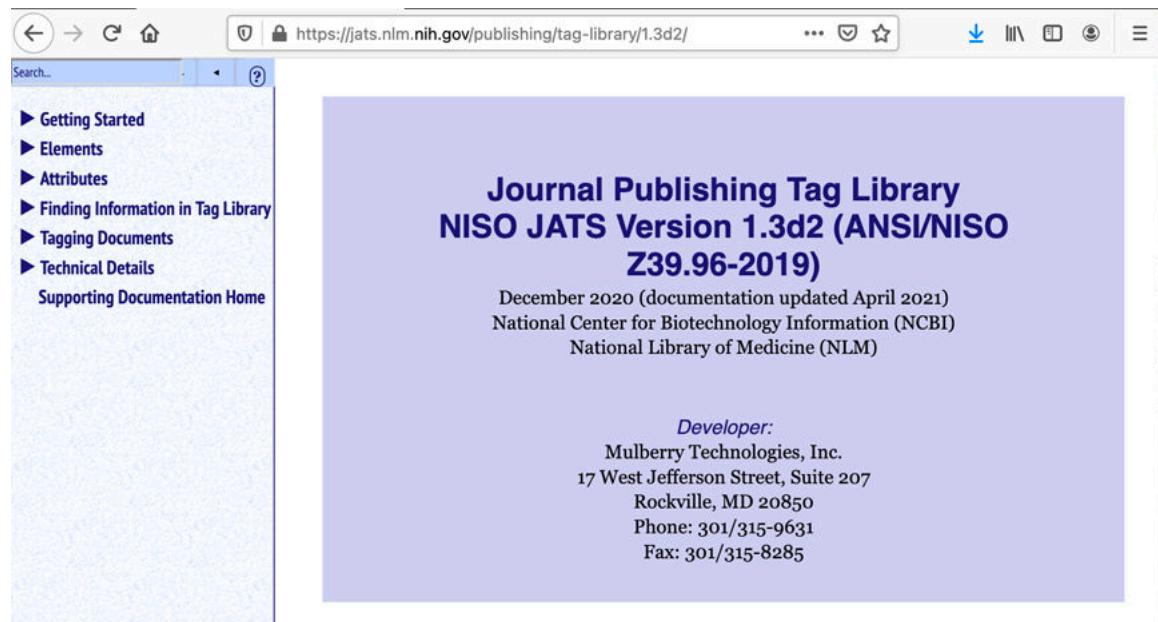


Figure 3. Article Authoring (Pumpkin) Tag Library



4.1. Navigation

4.1.1. The NAVBAR

On the left side of each HTML page is the Navigation Bar (NAVBAR). The NAVBAR shows an outline of the main parts of the Tag Library. Items prefaced by a right-facing arrow may be expanded (like an outline). Click either on the right-facing arrow or on the title of the section to both expand that portion of the navigation bar and open the introduction to that section. Similarly, click on a subsection heading to open that subsection in the main panel.



4.1.2. Opening and closing subsections on a page

Small diamonds on the top right of each page provide accordion folding inside the body of most pages:



The hollow diamond () opens all subheadings on the page.



The solid diamond () closes all subheadings *that can be closed*. Best Practice sections are left open and do not close.

◇ The diamonds are “sticky”, meaning that the next page you open will come up with the last setting you used. This is viewed as an accessibility feature, set it and forget it.

4.1.3. Cross references

Throughout the text, live internal cross references (such as element and attribute names) appear in color. Hovering over a cross reference to an element or attribute will show the narrative name of the element or attribute

**ection of the paper is summarized i
n being summarized. Such abstract
or the element <abstract> has be
l that most abstracts Abstract > much s**

- ◇ Cross references to elements are the element tag name (generic identifier), surrounded by delimiters (<abstract>). Almost all element names, including those in the Context and natural language Content Model Descriptions are live links to the specific element page.
- ◇ Attribute references appear as the attribute name, prefaced with an at sign (@specific-use). These link directly to the specific attribute page.
- ◇ Parameter Entity (PE) references are the name of the PE, prefaced by a percent sign and closed with a semicolon (%list.class;). These link directly to the parameter entity page.
- ◇ References to narrative sections of the Tag Library are preceded by the word “See”, and link to sections in the narrative text.
 - For a discussion on tagging processing alternatives, see [Tagging Alternative Versions](#).
 - See: [Hierarchy diagram - Alternatives For Processing](#)

4.2. Top-level element in the Schema

The Getting Started section at the top of the NAVBAR holds a very important piece of information, not just for beginning users but for all taggers, namely, what is the root element of this schema?

Figure 4. JATS root element



Here is the similar root element section for the STS (ANSI/NISO Z39.102-2017, STS: Standards Tag Suite;) Tag Library, showing multiple root elements possible:

Figure 5. STS has two possible root elements

Root Elements

NISO STS contains two root elements, either of which may be used as a document element to enclose a standards document:

- **Standard:** The `<standard>` element is used for standards and other standards-like documents. A `<standard>` may be at any stage in the standards and document life cycle, it may be developed by and published by one or more organizations, and it may be a simple short document or a very complex and/or long document. Although these Tag Sets were designed for normative standards, if the elements in this Tag Suite can be used to tag another type of standards document appropriately, using the `<standard>` element for a document such as, for example, a Guide or Handbook, is not misuse of the NISO STS.
- **Adoption:** In the situation where a completed standard is adopted by organizations that did not create or publish it, the `<adoption>` structure contains information about the adoption *and* the original standard. An adoption may include information on the organization(s) adopting the standard and include front matter (such as introductions) and back matter provided by the adopting organization(s). The `<adoption>` element can be used for nested adoptions and for adoptions of one or more `<standard>`s.

NISO STS 1.0 Tag Suite Components—Including TBX-TML

[Back to Top](#)

4.3. Elements Pages

There is one element page for each element in the Tag Set. Each element page begins the tag name (Generic Identifier), a descriptive natural-language name, and a description, all information that is shared with the JATS Standard. Descriptions are *not* dictionary definitions, but narrative text describing the element, as the name implies.

The identification information may be followed by sections such as Usage/Remarks and Related Elements, which are not part of the JATS Standard.

4.3.1. Usage/Remarks

This section provides additional information about the element, explanations of similar or contrasting elements, or instructions for element usage. (See also Related Elements described below.)

Figure 6. Usage/Remarks for `<abstract>`

`<abstract>` Abstract



Summarized description of the content of a document or document component.

Usage/Remarks

Many journal publishers request an abstract that is a very short summary of the major findings or conclusions of an article and limit this abstract to a paragraph or two. Some publishers require “long” or “summary” abstracts in which each section of the paper is summarized in a separate abstract section that has the same title as the article section being summarized. Such abstracts may be extensive, incorporating figures and tables. While the model for the element `<abstract>` has been made flexible enough to allow for these titled sections, it is expected that most abstracts will be much simpler and will contain one or more paragraphs.

- ▶ **Multiple Abstracts**
- ▶ **Locations of Abstracts**
- ▶ **Accessibility**

▼ Related Elements

The element `<trans-abstract>` contains another version of the abstract, one translated into a language other than that of the original publication.

This section also contains notes, such as Historical Notes on earlier JATS versions; Conversion Notes and Technical Notes to people who are mapping between documents tagged according to this Tag Set and those tagged according to other tag sets; and Implementor's Notes, which are directed to people building custom tag sets.

Figure 7. Usage Remarks for <year>

Usage/Remarks

The `@calendar` attribute can be used to indicate the calendar (e.g., Gregorian, Thai Buddha, or Japanese) of the given year.

In addition to being used for the year of publication, the `<year>` is also used to record “historical” events in the publishing cycle, for example, the year the document was accepted or last updated.

The `<year>` element is used in two contexts: as a part of the metadata concerning the article itself, and as part of the description of a cited work inside a bibliographic reference (`<element-citation>` or `<mixed-citation>`).

Best Practice

When possible, the year should be expressed as a 4-digit number, for example, “1776”, “1924”, or “2015”.

► Related Elements

Some “Usage/Remarks” sections are very complex, containing folded (collapsed) subsections, which the user will need to open to read. The “Best Practice” sections are highlighted with a light colored background, and such sections will be open (not collapsed) on opening.

Figure 8. Usage Remarks for <article-id>

<article-id> Article Identifier

Unique external identifier assigned to an article.

Usage/Remarks

Multiple Identifiers

There may be many identifiers for a single article, and an identifier may be a unique identifier in some systems. The content for this element may be assigned by a processor, for example. Examples of such identifiers include the article number, etc.

Attribute Best Practice

Type of Identifier Best Practice

Best Practice is to use the `@pub-id-type` attribute to indicate whether the identifier is a publisher's identifier, a DOI, a PMID, an aggregate identifier, or an identifier assigned by an indexing or hosting service, or similar. (See [@pub-id-type](#))

Authority for Identifier Best Practice

The attribute `@assigning-authority` should name the organization that issued the identifier or administers the identifier (such as CrossMark).

► External Identifier

► Conversion Note

► Historical Note

4.3.2. Attributes on an element page

All of the attributes available on an element are listed under the “Attributes” heading. For each attribute, the XML name of the attribute is shown, with the textual name of the attribute available as hover text. If the attribute is required, has a default value, or has a fixed value, then that is noted after the name. Each attribute is linked to its description in the Attribute Section of the Tag Library.

Attributes are grouped so that those most likely to be of interest are shown first, followed by groupings of:

- ◊ “Base Attributes” (common to all JATS elements),
- ◊ “Linking Attributes” (used to make an element into a live link),
- ◊ “Namespaces” (which are technically pseudo-attributes rather than attributes but which look and act a lot like attributes), and
- ◊ “Miscellaneous non-JATS-specific Attributes” (which are also technically pseudo-attributes rather than attributes but which look and act a lot like attributes).

Figure 9. Attributes (and pseudo-attributes) for the <article> element

▼ Attributes

```

article-type
dtd-version
specific-use

Base Attributes
id
xml:base
xml:lang (default = en)

Namespaces
xmlns:ali (fixed value = http://www.niso.org/schemas/ali/1.0/)
xmlns:mml (fixed value = http://www.w3.org/1998/Math/MathML)
xmlns:xlink (fixed value = http://www.w3.org/1999/xlink)
xmlns:xsi (fixed value = http://www.w3.org/2001/XMLSchema-instance)

Miscellaneous non-JATS-specific Attributes
xsi:noNamespaceSchemaLocation

```

Figure 10. Attributes for the <supplementary-material> element

▼ Attributes	
content-type	
hreflang	
mime-subtype	
mimetype	
orientation (default = portrait)	
position (default = float)	
specific-use	
Base Attributes	
id	
xml:base	
xml:lang	
Linking Attributes	
xlink:actuate	
xlink:href	
xlink:role	
xlink:show	
xlink:title	
xlink:type	
Namespaces	
xmlns:xlink	

4.3.3. Models and Context

The “Models and Context” section contains two very different type of information: where an element may be used (context) and what an element may contain (what may be inside the element). Under the heading “May be contained in”, there is a list of all of the JATS elements that may include the given element as a direct child.

Figure 11. Where may <abstract> be used?

▼ May be contained in

```
<ack>, <app-group>, <article-meta>, <chem-struct-wrap>, <disp-formula>, <disp-formula-group>, <fig>, <fig-group>, <front-stub>, <graphic>, <media>, <sec-meta>, <statement>, <supplementary-material>, <table-wrap>, <table-wrap-group>
```

The second kind of information is the content model of the element, that is, what an element may contain. Content model information is available in 3 formats: hierarchy written as a bulleted list in natural language, the content model as given in the DTD for that element (typically useless because it is only parameter entities), and the DTD model after all the parameter entities have been expanded.

Figure 12. Content model of <answer-set>

▼ Description

The following, in order:

- <object-id> Object Identifier, zero or more
- <label> Label (of an Equation, Figure, Reference, etc.), zero or one
- <title> Title, zero or one
- <subtitle> Document Subtitle, zero or more
- <alt-title> Alternate Title, zero or more
- One or more of any of:
 - <answer> Answer to a Question
 - <p> Paragraph
 - <explanation> Explanation

▼ Content Model

```
<!ELEMENT answer-set %answer-set-model; >
```

▼ Expanded Content Model

```
((object-id)*, label?, title?, subtitle*, alt-title*, (answer | p | explanation)+)
```

4.3.4. Tagged Samples

Almost all the elements and some of the attributes have sample document fragments, tagged in XML. Some elements have multiple samples, showing the element used in a variety of circumstances. Each sample is described with a title (and some with captions as well) to direct the user to the sample(s) which meet their needs.

Figure 13. Tagged samples for <article>

▼ Tagged Samples

- ▶ Article with front matter, body, and back matter

- ▼ Samples article attributes

```
<article
    article-type="research-article"
    specific-use="export-for-online"
    xml:lang="en"
    xmlns:mml="http://www.w3.org/1998/Math/MathML"
    xmlns:xlink="http://www.w3.org/1999/xlink" dtd-version="1.3d2"
    xmlns:ali="http://www.niso.org/schemas/ali/1.0/">
    <front>...</front>
    <body>...</body>
    <back>...</back>
</article>
```

Figure 14. Tagged samples for <affiliation>

▼ Tagged Samples

- ▶ Inside contributor
- ▶ End of contributor group
- ▶ Naming affiliated institution and its identifier
- ▶ Inside person group in element citation
- ▼ Affiliation name in Japanese and English

Attributed to the NISO JATS Standard, Version 1.3d2, 2017-03-01, https://jats.nlm.nih.gov/1.3d2/

4.3.5. Other sections on the Element Pages

There are other optional sections that may appear on an Element Page:

Related Elements

Contains information about elements associated with or confused with the current element. For some complex structures, such as two-part definition lists (<def-list>), this section names all the elements that may make up such a list (title, headings, terms, definitions, etc.) and describes the structure.

Related Resources

Provides pointers both to external information sources and to other parts of the Tag Library (such as Hierarchy Diagrams and Essays) that contain information relevant to this element.

4.4. Attribute Pages

There is one element page for each element in the Tag Set. Each attribute page begins with the attribute name used in the XML files, a longer descriptive name, and a description, all information that is shared with the JATS Standard. Descriptions are *not* dictionary definitions, but narrative text describing the attribute, as the name implies.

A “Usage and Remarks” section often provides additional information on how to use the attribute, sometimes including “Best Practice” suggestions.

Figure 15. Attribute page for @object-id

@object-id Object Identifier



Identifier of an object (for example, a table, figure, or sidebar) within a separate document that is the target of the `<related-object>` element.

Usage/Remarks

Used along with `@object-id-type` to identify the object within a document that is the target of the `<related-object>` element. The `@object-id` might contain a DOI as content, and the `@object-id-type` should indicate that this identifier is a DOI. The `@object-id` also might contain the XML ID of the target object.

Best Practice

For each source, document, and object, an identifier should be specified on the corresponding attribute. For any related-object link, the `@source-id` attribute points to the largest publishable unit, for example, an entire book. The `@document-id` attribute points to a major component of the source, for example, a chapter, front matter section (Preface), or back matter section

Figure 16. Attribute values for @assigning-authority

▼ OPTIONAL on many elements; click for list and usage

```
<article-id>, <article-version>, <award-id>, <compound-kwd>, <compound-subject>
<contrib-id>, <custom-meta>, <ext-link>, <extended-by>, <institution-id>, <isbn>
<issn>, <issn-l>, <issue-id>, <journal-id>, <kwd>, <kwd-group>, <nested-kwd>,
<object-id>, <pub-date>, <pub-id>, <resource-id>, <restricted-by>, <role>, <self-uri>,
<subj-group>, <subject>, <uri>, <volume-id>
```

Value	Meaning
Text, numbers, or special characters	The name of the organization assigning the identifier, such as “ORCID” or “ISNI”
Restriction	<code>@assigning-authority</code> is an optional attribute; there is no default. The attribute is only used when the authority is known.

Suggested usage

This attribute may take any text value, but it should name the authority, for example, (not complete list, just exemplars):

arxiv arXiv archive of electronic preprints

crossref Crossref

doaj Directory of Open Access Journals

figshare Figshare data repository

GenBank NIH genetic sequence database

mr Mathematical Reviews (MR)

PDB Protein Data Bank

Figure 17. @toggle is an attribute with some #FIXED values:

@toggle Toggle Switch

Specifies if the styling of the element it modifies should act as a toggle-switch. When the toggle-switch is on, the textual content of the element will always be visually set apart from its context. When the switch is off, the rendition is fixed to the style requested and will not change based on context.

- ▶ **OPTIONAL (defaults to yes) on element: <italic>**
- ▶ **OPTIONAL (defaults to no) on element: <roman>**
- ▶ **OPTIONAL on many elements; click for list and usage**

Some attributes are allowed on more than one element, and may have different meanings, default values, and suggested values depending on the element on which they are used. When an attribute is used in more than one element, there is a separate section for each group of elements that share the same usage. The section head identifies whether the attribute is optional, is required, has a default value (if any), and then names the other elements to which this attribute applies. If there are too many applicable elements to list in the heading, a message says “click for list and usage”

Figure 18. Other variations in attribute values (@rid)

@rid Reference to an Identifier

Value of the identifier of an associated element; used for linking related elements (for example, <term> to a <def>, <bio> to a <contrib>).

- ▶ **OPTIONAL on elements: <milestone-end>, <milestone-start>**
- ▶ **REQUIRED on element: <index-term-range-end>**
- ▼ **OPTIONAL on many elements; click for list and usage**
`<aff>, <author-notes>, <award-group>, <award-id>, <bio>, <contrib>, <def>, <funding-source>, <funding-statement>, <named-content>, <see>, <see-also>, <sig>, <sig-block>, <support-description>, <support-source>, <term>, <xref>`

Value	Meaning
One or more identifiers (IDREFS)	Points to one or more existing identifiers.
Restriction	@rid is an optional attribute; there is no default.

Tagged Samples are provided for some attributes, though not all.

Figure 19. Tagged sample for @journal-id

@journal-id Journal Identifier of a Related Article

Identifier for the journal that contains the related article.

Usage/Remarks

The @journal-id attribute can be used, along with the @vol, @issue, and @page attributes, to provide the metadata for the related article.

- ▶ **OPTIONAL on element: <related-article>**

▼ Tagged Sample

Giving the journal identifier for an Addendum (a <related-article>)

```
...
<related-article elocation-id="052207" ext-link-type="uri"
    related-article-type="addendum" xlink:href="7596428"
    journal-id="MULB" journal-id-type="publisher-id" vol="281"
    issue="318" page="582"/>
...
```

4.5. Finding Information Pages

There are two subsections within this section:

Index

Provides pointers from both preferred terms (such as element and attribute names) and non-preferred terms (such as synonyms of common JATS elements, such as “author” “See <contrib>” for terms used throughout the Tag Library.

Element Context Table

A consolidated alphabetical of the context information that appears on each element page. The table provides a listing of where each element may be used, that is, all the contexts in which this element may occur.

Index — The index includes tag names, element names, attribute names, parameter entity names, elements from other vocabularies that may be familiar to JATS users, and synonyms to many JATS element name.

Figure 20. How do I tag the article author in JATS?

```

author
  see <attrib>
  use <contrib>
  see @contrib-type
author, corresponding
  see <corresp>

```

Figure 21. Why doesn't JATS have a “quote” element?

Quote, Displayed
Quote, Displayed Model parameter entity
 see %disp-quote-model;

Element Context Table — The Context Table describes where each JATS element may be used. The table is formatted in two columns:

The first column

(“This Element”) names an element

The second column

(“May Be Contained In”) contains an alphabetical list of all the elements in which the element named in the first column element may occur.

The letters of the alphabet in the NAVBAR and at the top of each page are links directly to the elements which start with that letter.

Figure 22. Context Table entry for <disp-quote>

```

<disp-quote> <answer>, <app>, <app-group>, <bio>, <body>, <boxed-text>,
  <disp-quote>, <explanation>, <fig>, <glossary>,
  <license-p>, <named-content>, <notes>, <option>, <p>,
  <question>, <question-preamble>, <ref-list>, <sec>,
  <styled-content>, <supplementary-material>, <table-wrap>, <td>, <th>

```

Figure 23. Context Table entry for <boxed-text> and <break>

<boxed-text>	<answer>, <app>, <app-group>, <bio>, <block-alternatives>, <body>, <boxed-text>, <disp-quote>, <explanation>, <floats-group>, <glossary>, <license-p>, <named-content>, <notes>, <option>, <p>, <question>, <question-preamble>, <ref-list>, <sec>, <styled-content>
<break>	<aff>, <alt-title>, <article-title>, <chem-struct>, <disp-formula>, <product>, <sig>, <sig-block>, <subtitle>, <td>, <th>, <title>, <trans-subtitle>, <trans-title>

Now that seems odd, why is <break> allowed in so few places? If we look at the element page for <break> we find out, as part of the Best Practice information:

Figure 24. Element Page entry for <break>

<break> Line Break



An explicit line break in the text.

Usage/Remarks

This element is restricted to a very few contexts, for example, inside <title>s and inside table cells (<td>s and <th>s).

Authoring/Best Practice

Usage is discouraged. The intent of this Tag Set is not to capture the “look-and-feel” of a document.

- ▶ **Attributes**
- ▶ **Models and Context**
- ▶ **Tagged Sample**

4.6. Stupid navigation tricks

This section collects odd information about navigation that every user ought to know about:

- ◊ Question Mark
- ◊ Hiding the NAVBAR
- ◊ Bringing back the NAVBAR
- ◊ Closing open sections
- ◊ The Search Bar

Question mark (?) — Shows the help for the NAVBAR

?

Use ← to hide the navigation sidebar.

Use ← to unexpand headings in the sidebar.

Search box instructions:

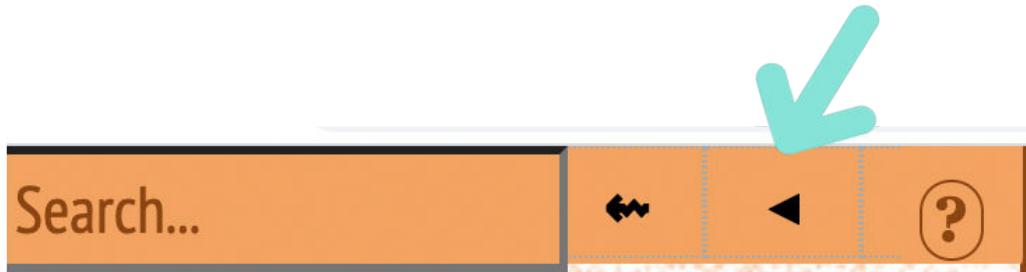
Use < to search for an element.

Use & to search for an attribute.

Use % to search for a parameter entity.

Or just type for a substring search.

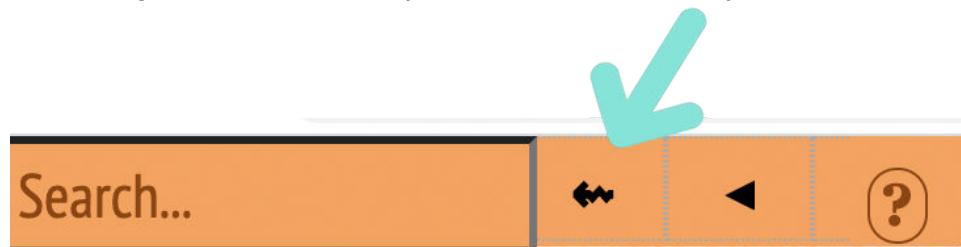
Hide NAVBAR — The solid right-facing arrow hides the NAVBAR.



Bring back NAVBAR — While (once the NAVBAR is hidden), the solid left-facing arrow will bring it back.



Closing all open sections — When there are a lot of open sections on your screen (element sections, attribute sections, essays), it can be painful to scroll to each open section and collapse it. The JATS Tag Libraries *almost* have a solution for that. The small squiggle arrow to the right of the search box is a collapse function. Click it to collapse all open sections, **except the one you are inside**. So, if you have all the elements open, and several essays open, and all the attributes open, and you are looking at an attribute page, when you click the squiggle the elements and essays will close and the attribute section will stay expanded. This is a known bug, that will one day be fixed. Until the bug is gone, the hack is to click on “Getting Started” section, then click the squiggle arrow, which closes all but “Getting Started”, which is very short and can be manually closed.



Search Bar — If 1) you are reading the Tag Library over the web (not a local copy) and 2) your browser and system support this feature, you can search within the Tag Library using the Search bar, found at the very top left of the screen.

There are five types of searches:

<tag-name>

Enter a less-than-sign ("<") followed by the element name to search for an element

&attribute-name

Enter an AT-sign ("@") followed by the name of an attribute to search for an attribute

%pe-name

Enter a percent-sign ("%") followed by the name of a parameter entity to search for a parameter entity

As you type in the Search Bar, possibilities will appear in a list below, and you can choose one.



Remember, click the Question Mark ("?") for hints on using the search bar.

5. The rest of the Tag Library

5.1. Getting Started (for the casual user and newbies)

The US National Library of Medicine conducted some usability studies on the JATS homepages and the Tag Library documentation. Users had a lot of good suggested, many of which the latest Tag Libraries implement. But the only users interviewed were experienced ones, and they were unanimous on the recommendation to remove "all that material junking up the NAVBAR", that is, the material now (hidden) under the section "Getting Started".

The Getting Started section is just one of the features aimed at the casual or beginning user, which include:

- ◊ Getting Started section
- ◊ Introductions (everything has one)
- ◊ Index with lots of *use* and *see* entries
- ◊ Hierarchy diagrams
- ◊ Content models written in natural language with words like "followed by" and "optional".

Figure 25. Getting Started Section

5.1.1. Introductions

Almost all sections in the Tag Library have Introductions. To find the introduction to a section, click on the *name of the section* rather than the arrow before the section name. The subsections will open up, but if there is an Introduction it will fill the main panel. At one of the JATS Standing Committee meetings it was revealed that the majority of members had no idea the introductions were there, or how to find them, why they sometimes suddenly appeared, or why you would want to find them.

Figure 26. Introduction to the Elements Pages

Introduction to Elements

Each element in the Tag Set and the XHTML-inspired table model has a separate Tag Library describing the OASIS Exchange CALS Tag Library at <https://jats.nlm.nih.gov/options/OASIS/tag-library/19990311/>.

The elements in this Tag Library are described in alphabetical order by tag type names. The tag name is the shorter machine-readable name used in DTDs and schemas, and (typically) by software; for example, the tag for a named Paragraph.

Content of an Element Page

Each element is described in a single page that displays:

- The element's tag name followed by its longer descriptive name
- Sections describing aspects of the element and its usage

The sections within the page always appear in the following order:

Description Provides a narrative description of the element. (untitled) This is not intended to provide information about the rationale for the element.

Usage/Remarks Provides additional information about the instructions for element usage. (See also Remarks.) Best Practice notes are identified with a heading that has a colored background for emphasis.

More complex “Usage/Remarks” sections may be needed for navigating in this document, including opening and closing tags. Conversion Notes and Technical Notes are provided for people who are mapping between documents tagged with different tag sets; building conversion software for this Suite; or producing products based on the general reader will need.

Authoring Notes are usage instructions aimed at people who are writing content to this Tag Set.

Implementor’s Notes are instructions written for people who are implementing the Suite based on the Suite.

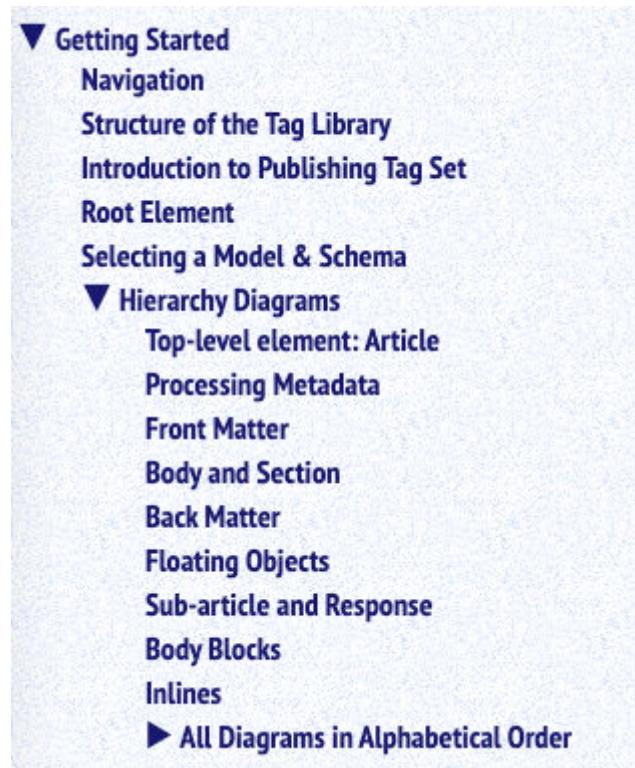
Related Elements Contains information about elements associated with this element, such as complex structures, such as two-part definitions.

5.1.2. Hierarchy Diagrams

Hierarchy diagrams are graphical representations of the structure of elements, for elements that have complex structure. An element with a simple model, such as an “or” list (that is, where any of a number of elements may occur inside it in any order), for example Paragraph (<p>), is not illustrated with a hierarchy diagram. An element that must have one of these, followed by any of this list of things, followed be either this or that, will be illustrated with a hierarchy diagram.

All of the Hierarchy diagrams may be reached through the Getting Started Section. An individual diagram for a specific element can be reached through the Related Resources subsection on the element page:

Figure 27. Finding the hierarchy diagrams



The first page of the Hierarchy Diagrams section describes the symbols used in the diagrams

If a box has an element name, then symbols at the left end of the box indicate whether that element is required and/or repeatable. These symbols are called “occurrence indicators”:

- | | |
|------------------|--|
| ? | item is optional (zero or one) |
| * | item may occur any number of times (zero or more) |
| + | item must occur at least once, but may occur any number of times (one or more) |
| Element | vertical bar on the left of the box: item is the “document element”: the top-most element, such as <article> or <book> and is therefore required |
| (no symbol) | item must occur exactly once |

The symbols at the right end of a box have these meanings:

- | | |
|------------------|--|
| ~ (a tilde) | the item may take one or more attributes |
| Element | vertical bar on the right of the box: the item is expanded elsewhere (for example, if an element is permitted in more than one place in an element, it is expanded only once.) |

Key to the Near & Far Diagrams

Element Occurrences

Element	Required
? Element	Optional. May occur zero or one times.
+ Element	Required, repeatable. May occur one or more times.
* Element	Optional, repeatable. May occur zero or more times.

Additional Symbols Associated with Elements

Element	Element content is expanded elsewhere in the diagram.
Element ~	The element has attributes.
...	One or more elements collapsed for clarity
I Element	Element is the root element of the model.
≡	Text, numbers, and special characters

Grouping Elements

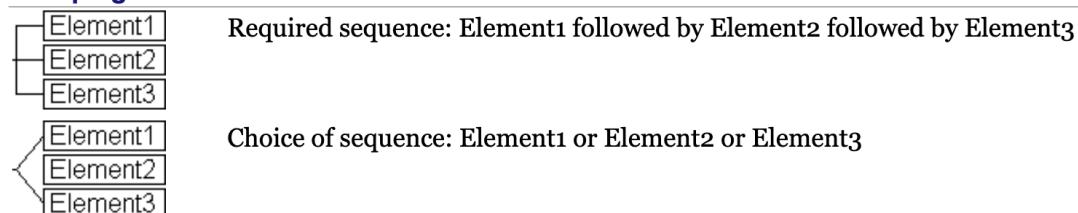


Figure 28. The hierarchy diagram for Article (<article>)

Top-level element: Article

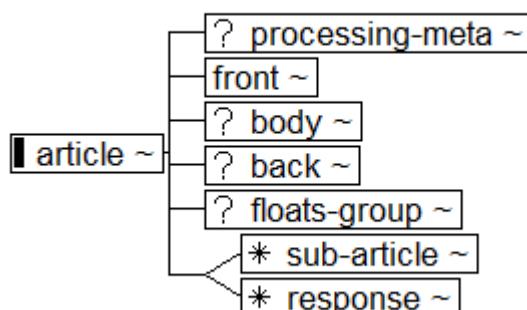


Figure 29. The hierarchy diagram for Back Matter (<back>)

Back Matter

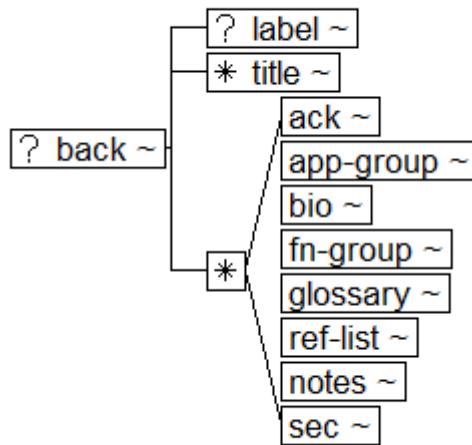
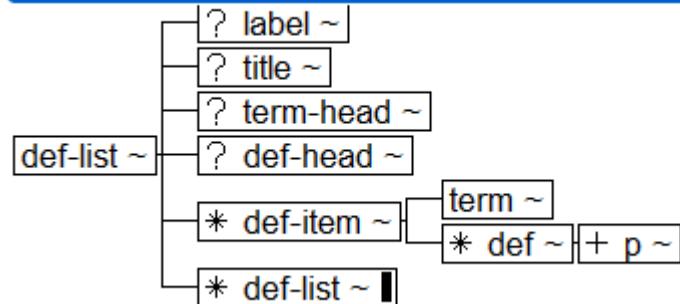


Figure 30. The hierarchy diagram for Definition List (<def-list>)

▼ Definition List



5.2. For the Frequent user

Many Tag Library features are aimed directly at the more frequent user:

- ◊ Search box (why scroll?)
- ◊ Collapse, open all (diamonds are your friend)
- ◊ Content models in DTD syntax (more compact)
- ◊ Best Practice remarks open with colored background
- ◊ Multiple labeled tagged samples for most elements
- ◊ Full Article Samples
- ◊ Common Tagging Practice essays

5.2.1. Multiple labeled samples per element

Most elements and some attributes are given more than one tagged sample (some have over a dozen). To differentiate the samples and lead a user more directly to the one that matches their situation, each sample has a heading that *tries* to explain why the sample

was included, what it is trying to illustrate. One of the more common complaints of JATS users is that the sample they want or need is not present.

Figure 31. Samples for <institution-id>

Tagged Samples

- ▶ In <institution-wrap> inside <aff>
- ▶ Funding metadata with vocabulary attributes
- ▼ One <institution-wrap>, multiple identifiers

```
...
<funding-group>
  <award-group>
    <funding-source>
      <institution-wrap>
        <institution content-type="laboratory">Massachusetts Green High Performance
          Computing Center</institution>
        </institution-wrap>, operated by

      <institution-wrap>
        <institution-id institution-id-type="Ringgold">1846</institution-id>
        <institution-id institution-id-type="ISNI">0000 0001 2170 1429</institution-id>
        <institution content-type="university">Boston University</institution>
      </institution-wrap>

      <institution-wrap>
        <institution-id institution-id-type="Ringgold">1812</institution-id>
        <institution content-type="university">Harvard University</institution>
      </institution-wrap>,
```

Samples are maintained as valid XML fragments that are parsed for validity each time the DTD changes. Processing instructions inside the fragments are used to create the bold focus. Some samples are nearly full documents, others are as small as possible:

```
<!DOCTYPE body PUBLIC
"-//NLM//DTD JATS (Z39.96) Journal Publishing DTD v1.3d2 20201130//EN" "JATS-journalpublishing1-3d2.dtd">
<?mtl hellip?>
<body><?mtl /hellip?>
<p>And I can say without fear of contradiction
<abbrev alt="Wink smile">);)</abbrev> that this
political process is without flaw.</p>
<?mtl hellip?></body><?mtl /hellip?>
```

5.2.2. Full article samples

The completely tagged article sample is also aimed at the more frequent user. It illustrates one way an entire JATS document can be put together. Full samples are not available for the Archiving Tag Set (Green) because, as an archival format that must accept whatever is sent to the archive by publishers, there is no such thing as a typical Archiving sample.

Figure 32. The Full Article Sample section

Full Article Samples

Samples of two complete journal articles are provided. For each sample, a PDF file showing the published format of the article is provided as well as an XML file containing the content of the article in XML according to the Journal Publishing Tag Set. The samples are:

- From BMJ:
 - Article in PDF: [click here for PDF file](#)
 - Article in XML (with .txt filename extension): [click here for XML file](#)
 - Article in XML (with .xml filename extension, for easy reading in XML-smart browsers): [click here for XML file](#)
- From PNAS:
 - Article in PDF: [click here for PDF file](#)
 - Article in XML (with .txt filename extension): [click here for XML file](#)
 - Article in XML (with .xml filename extension, for easy reading in XML-smart browsers): [click here for XML file](#)

Figure 33. Document shown as PDF

Primary care

Evolving general practice consultation in Britain: issues of length and context

George K Freeman, John P Horder, John G R Howie, A Pali Hungin, Alison P Hill, Nayan C Shah, Andrew Wilson

Centre for Primary Care and Social Medicine, Imperial College of Science, Technology and Medicine, London W6 8RP
 George K Freeman, professor of general practice

Royal College of General Practitioners, London SW7 1PU
 John P Horder, past president
 Nayan C Shah, general practitioner

Department of General Practice, University of Edinburgh, Edinburgh EH8 9DX
 John G R Howie, emeritus professor of general practice

Centre for Health Studies, University of Durham, Durham DH1 3HN
 A Pali Hungin, professor of general practice

Kilburn Park Medical Centre, London NW6
 Alison P Hill, general practitioner
 Department of

In 1999 Shah¹ and others said that the Royal College of General Practitioners should advocate longer consultations in general practice as a matter of policy. The college set up a working group chaired by A P Hungin, and a systematic review of literature on consultation length in general practice was commissioned. The working group agreed that the available evidence would be hard to interpret without discussion of the changing context within which consultations now take place. For many years general practitioners and those who have surveyed patients' opinions in the United Kingdom have complained about short consultation time, despite a steady increase in actual mean length. Recently Mechanic pointed out that this is also true in the United States.² Is there any justification for a further increase in mean time allocated per consultation in general practice?

We report on the outcome of extensive debate among a group of general practitioners with an interest in the process of care, with reference to the interim findings of the commissioned systematic review and our personal databases. The review identified 14 relevant papers.

Longer consultations: benefits for patients

The systematic review consistently showed that doctors with longer consultation times prescribe less and offer more advice on lifestyle and other health promoting activities. Longer consultations have been significantly

Summary points

Longer consultations are associated with a range of better patient outcomes

Modern consultations in general practice deal with patients with more serious and chronic conditions

Increasing patient participation means more complex interaction, which demands extra time

Difficulties with access and with loss of continuity add to perceived stress and poor performance and lead to further pressure on time

Longer consultations should be a professional priority, combined with increased use of technology and more flexible practice management to maximise interpersonal continuity

Research on implementation is needed

Context of modern consultations

Shorter consultations were more appropriate when the population was younger, when even a brief absence from employment due to sickness required a doctor's note, and when many simple remedies were available

Figure 34. Document shown as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE article PUBLIC \
"-//NLM//DTD JATS (Z39.96) Journal Publishing DTD v1.3d2 20201130//EN"
 "JATS-journalpublishing1-3d2.dtd">
<article
  article-type="research-article"
  dtd-version="1.3d2"
  xml:lang="en"
  xmlns:mml="http://www.w3.org/1998/Math/MathML"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
<processing-meta
  tagset-family="jats"
  base-tagset="publishing"
  mathml-version="2.0"
  table-model="xhtml"/>
<front>
<journal-meta>
<journal-id journal-id-type="pmc">bmj</journal-id>
<journal-id journal-id-type="pubmed">BMJ</journal-id>
<journal-id journal-id-type="publisher">BMJ</journal-id>
<issn>0959-8138</issn>
<publisher>
<publisher-name>BMJ</publisher-name>
</publisher>
</journal-meta>
<article-meta>
<article-id pub-id-type="other">jBMJ.v324.i7342.pg880</article-id>
<article-id pub-id-type="pmid">11950738</article-id>
<article-version vocab="JAV"
  vocab-identifier="http://www.niso.org/publications/rp/RP-8-2008.pdf"
  article-version-type="VoR"
  vocab-term="Version of Record">version-of-record</article-version>
```

Figure 35. Document shown as syntax-colored and indented XML

```

▼<article xmlns:mml="http://www.w3.org/1998/Math/MathML" xmlns:article" dtd-version="1.3d2" xml:lang="en">
  <processing-meta tagset-family="jats" base-tagset="publishing">
    <front>
      <journal-meta>
        <journal-id journal-id-type="pmc">bmj</journal-id>
        <journal-id journal-id-type="pubmed">BMJ</journal-id>
        <journal-id journal-id-type="publisher">BMJ</journal-id>
        <issn>0959-8138</issn>
      <publisher>
        <publisher-name>BMJ</publisher-name>
      </publisher>
    </journal-meta>
    <article-meta>
      <article-id pub-id-type="other">jBMJ.v324.i7342.pg880</article-id>
      <article-id pub-id-type="pmid">11950738</article-id>
      <article-version vocab="JAV" vocab-identifier="http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3247342/record">
      <article-categories>
        <subj-group>
          <subject>Primary care</subject>
        <subj-group>
          <subject>190</subject>
          <subject>10</subject>
          <subject>218</subject>
          <subject>219</subject>
          <subject>355</subject>
          <subject>357</subject>
        </subj-group>
      </article-categories>
      <title-group>
        <article-title>Evolving general practice consultation in</article-title>
      </title-group>
    </article-meta>
  </front>
</article>

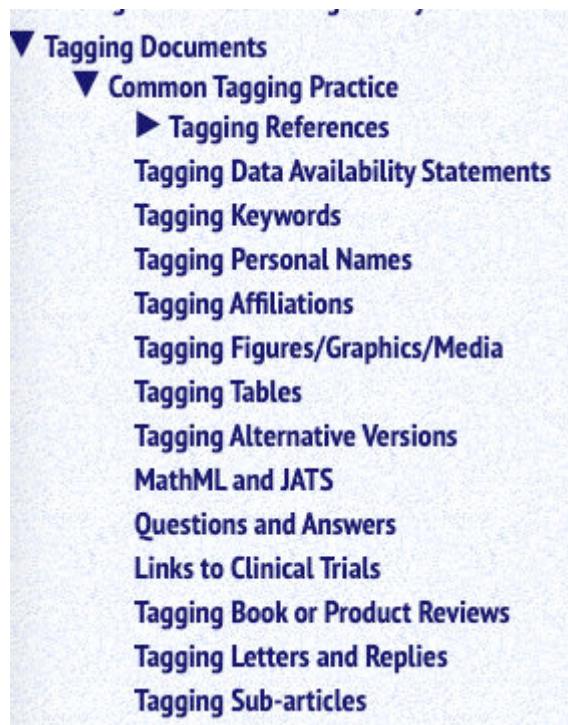
```

5.2.3. Common Tagging Practice essays (*almostutorials*)

The Tag Library has numerous short essays on how to tag various structures and use various attributes, mostly aimed at the experienced user, but suitable for the casual user. These essays are the closest JATS comes to tutorial material in the Tag Library, and, if the were read, they would probably prove helpful.

JATS is a descriptive not a prescriptive tag set, so the Tag Library does not include long essays on best practice or any Best Practice Recommendations. That type of guidance is provided by outside groups such as JATS4R [JATS4R]. The “Common Tagging Practice” section describes ways (sometimes many alternatives) that JATS *might be used* to tag documents and information on JATS and accessibility.

Figure 36. Subjects covered in Common Tagging Practice



5.2.3.1. Tagging keywords

Keywords are words and phrases used to name an article's or section's key concepts for search and retrieval purposes. Typically an author, publisher, or indexing service will assign a small number of key terms to expand lookup beyond full text, to point up the most important topics described in an article, or to map an article to a taxonomy.

JATS can describe simple keywords, compound keywords (code and text, abbreviations and expansions, etc.), nested keywords, and more. This essay describes how to tag them all, with samples for each type.

The last section of Tagging Keywords describes how to use the vocabulary attributes to encode that a keyword or group of keywords contains terms from a thesaurus (ontology, taxonomy, term-list, vocabulary, industry glossary, thesaurus, or other known term source), providing additional semantics for the term.

Figure 37. Selections from Tagging Keywords essay

▼ Tagging Complex/Compound Keywords

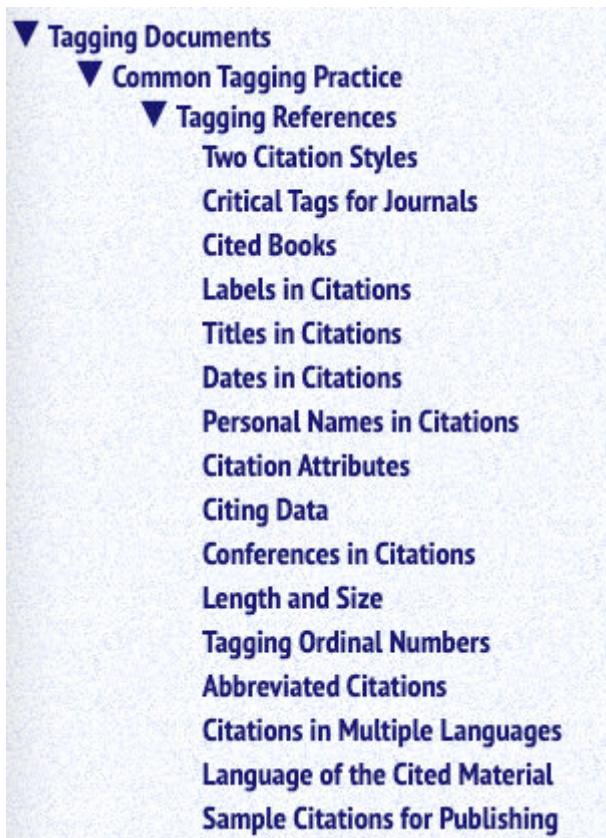
Keywords can possess an internal structure of their own; for example, a keyword may include both a textual phrase and its corresponding code ("863 Icelandic sagas"). Many styles of such compound keywords can be handled in this Tag Set with the `<compound-kwd>` element, which is modeled as a series of repeatable parts (`<compound-kwd-part>`). These parts can differentiate a text/code pair, divide a coded keyword into multiple code segments, describe a hierarchy, and name a variety of other compound structures. The `@content-type` attribute on the `<compound-kwd-part>` element is used to name each part, describe the role it plays, or otherwise define how each part functions within the keyword as a whole.

- **Keywords with Codes**
- **Abbreviation and Expansion Keywords**
- **Tagging Nested or Hierarchical Keywords**
- **Keywords from a Formal Taxonomy**

5.2.3.2. Tagging references

A lot of real estate has been spent on the many different kinds of bibliographic references and how they might be tagged. Tagging References includes short essays on different types of references, such as journal article references, data citations, and citations to conferences, as well as extensive examples.

Figure 38. Essays included in Tagging References



5.2.3.3. Citing data as a reference

Current publishing practice is to cite data sources in much the same manner that articles and books are cited. Such citations may be part of a regular Reference List or listed separately in their own list, either at the back of the article or inside a Data Availability Statement. One of the reference sections discusses data citations.

Figure 39. Citing Data section overview

Citing Data



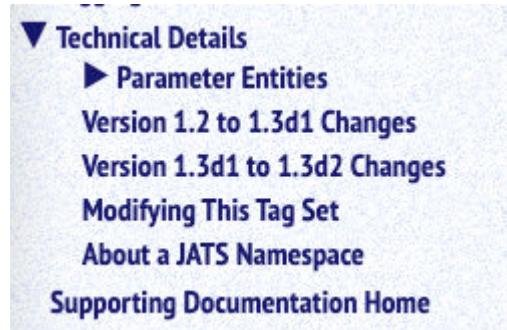
Current publishing practice is to cite data sources in much the same manner that articles and books are cited. Such citations may be part of a regular Reference List or listed separately in their own list, either at the back of the article or in a Data Availability Statement. (See [Data Availability Statement](#).)

- ▶ **Principles of Data Citation**
- ▶ **Data Citations in JATS**
- ▶ **Describing how the Data Files were Used**
- ▶ **Examples of Data Citations**

5.3. Technical Details

This is the section where the really geeky material is kept. This material is aimed at the software developer, the XMLer who need to make a JATS subset or superset, or the poor maintenance programmer who needs to keep a subset created (to fanfare) in 2001 fresh and running.

Figure 40. Content of the Technical Details Section

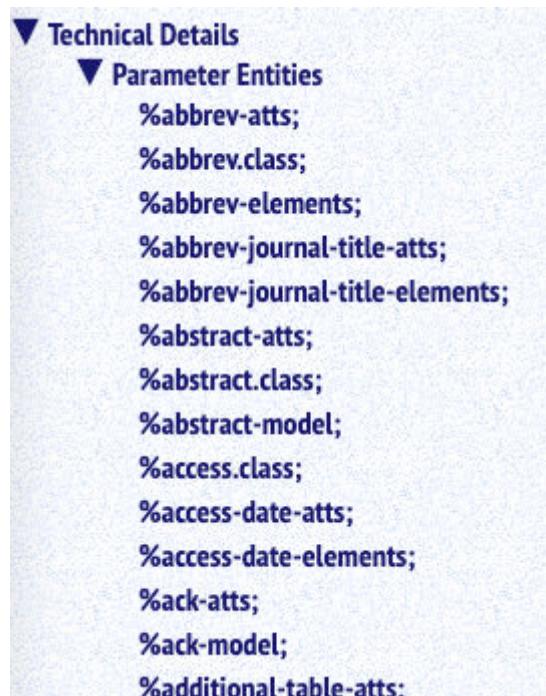


The screenshot shows a list of links under the heading 'Technical Details'. The links are:

- ▶ Parameter Entities
- [Version 1.2 to 1.3d1 Changes](#)
- [Version 1.3d1 to 1.3d2 Changes](#)
- [Modifying This Tag Set](#)
- [About a JATS Namespace](#)
- [Supporting Documentation Home](#)

Parameter Entity section — ' Parameter Entities, like elements and attributes, get a page apiece. There are many parameter entities, since JATS is maintained as a DTD and customized using Parameter entities. Each content model, each attribute list, each set of attribute values, and many grouping of elements are represented by a parameter entity.

Figure 41. Portion of the Parameter Entity section



The screenshot shows a list of parameter entity declarations under the heading 'Parameter Entities'. The declarations are:

- %abbrev-atts;
- %abbrev.class;
- %abbrev-elements;
- %abbrev-journal-title-atts;
- %abbrev-journal-title-elements;
- %abstract-atts;
- %abstract.class;
- %abstract-model;
- %access.class;
- %access-date-atts;
- %access-date-elements;
- %ack-atts;
- %ack-model;
- %additional-table-atts;

Figure 42. A Parameter Entity Page

%abstract-model; Abstract Model

Content model for the `<abstract>` element.

▼ Declaration

```
<!ENTITY % abstract-model
  "((%id.class;)*, label?, title?,
  (%just-para.class;)*, (%sec.class;)*)"      >
```

▼ Expanded Declaration

```
((object-id)*, label?, title?, (p)*, (sec)*)
```

Change History sections — Describes the ways in which each version of JATS is different from a previous version.

Figure 43. Citing Data section overview

Version 1.3d1 to 1.3d2 Changes



The ANSI/NISO Z39.96-2019: JATS 1.2 version is the latest ANSI/NISO standard JATS. This draft JATS 1.3d2 version (JATS Committee Draft) is fully backward compatible with JATS 1.0 (2012), JATS 1.1 (2015), JATS 1.2 (2019), and JATS 1.3d1. Any document valid to JATS 1.0, 1.1, 1.2, or 1.3d1 will be valid to the JATS 1.3d2 version of the same JATS Tag Set. Changes to JATS 1.3d1 to produce JATS 1.3d2 were made in response to user comments on the previous versions.

This 1.3d2 Committee Draft may not be stable and may change any time until the next version of the standard is approved by ANSI/NISO. There is no commitment that future drafts or future JATS standards will be compatible with this Committee Draft. Basing implementations on this draft may be premature.

- ▶ Catalog and Formal Public Identifiers in 1.3d2
- ▶ New Elements Added in 1.3d2
- ▶ Changes to Elements in 1.3d2
- ▶ New Attributes in 1.3d2
- ▶ Changes to Attributes in 1.3d2
- ▶ Element Description and Use
- ▶ New Parameter Entities in 1.3d2

Modifying This Tag Set — Provide information to developers who subset or superset JATS, by using the published JATS modules (unchanged!) to create JATS-based tag sets. Sections in this section describe the architecture and structure of a JATS tag set as well as the naming conventions used by JATS. By example, this section leads a developer through best practice in developing a new tag set based on JATS.

Figure 44. Subsections within Modifying This Tag Set

Modifying This Tag Set

This section contains implementor's instructions for using this making derivative tag sets based on this one.

- ▶ **First Steps: Using This Tag Library**
- ▶ **Modular DTD Design**
- ▶ **JATS and Linked Data**
- ▶ **How To Make New Tag Sets**
- ▶ **JATS Naming Conventions**
- ▶ **Modules in JATS Publishing and Suite**

6. Inconvenient truths and lessons learned

(in 40 years of documentation frustration)

First Lesson: Nobody reads the documentation! It's true, everyone praises the documentation, but almost no one reads it.

- ◊ Many people read only the examples (which they expect to be very specific to their situation!).
- ◊ Others complain about missing components that are present: "Why doesn't the JATS Tag Library have an index?" (Answer: It does.)
- ◊ Other read deep meaning into the simplest phrases.

Second Lesson: XML is not self-documenting, or "Why the Apple technique of no documentation does not work"

- ◊ Nobody uses the Index
- ◊ Assumptions are made based on element/attribute names
- ◊ To find the structure/semantics you want
 - ◆ scan the list of elements
 - ◆ pick one that sounds like what you need
 - ◆ if element allowed in your context, you're golden

There are many and varied reasons why semantic *guessing* might not work

- ◊ Names obscure (<csd> - as part of an address)
- ◊ Definitions do not adequately disambiguate (<pub-no> and <doc-no>)
- ◊ There is no match, you are looking for best approximation(<surname> and <given-names>)

To misquote Liam Quin only slightly: “ Documentation failures have been attributed to users assuming that they understood the content of an element when (perhaps) it had a misleading name.”

- ◊ Nobody reads the definitions
 - ◆ <on-behalf-of>
 - ◆ JATS <supplementary-material>

Inconvenient Truth #3: — Creation, maintenance and governance are heavy loads. We all realize that it has to be *somebody's* responsibility to write and maintain documentation. But I have a day job, don't you?

Inconvenient Truth #4: — Documentation costs money

- ◊ If a schema takes 2 weeks to write and test the documentation/samples will take 3-5 months to write and proof
- ◊ Tag sets change over time, *if they are being used*
- ◊ Documentation is harder to keep in sync than to create

Did anybody budget for maintaining this?

When should documentation be performed? — The only wrong answer: After everyone who created the document grammar has gone

Inconvenient Truth #5: How do you measure quality? — or, to put it another way: “What is good documentation?”

There are some vague general rules that apply to documenting a tag set:

- ◊ Don't describe processing, nothing is as ephemeral But that is the first thing people ask, “what does it do?”
- ◊ Don't describe format, unless that is the point (tables)

But that does not really answer the question. How do you measure good documentation? Good documentation may expose vocabulary flaws, but... the reverse does not hold. Programs can be tested, documentation not so much. Usability testing can help, with real users in real situations. So get your users involved!

7. Aside: Rome was not built in a day

JATS/NLM DTD has been around since ?1998? Documentation similar to what I have described today has been in use, by Mulberry, since the mid-80s. [Note: Only the automation is new.]

8. Discussion Topics

This section of the paper is the conclusion of the presentation and will be revised after I present the talk, and listen to the feedback and questions at the end. These are some of my preliminary questions. I'd like your feedback.

8.1. JATS-specific Questions

- ◊ What can JATS learn from the documentation for your tag set?

- ◊ What did you NOT like?
- ◊ Anything you've seen today where you said, "that's genius"?
- ◊ Anything you plan to copy?
- ◊ What is overkill?
- ◊ What would you change? Add?
- ◊ Is there anything in the documentation we could dump without loss? Introductions? Change Histories? Parameter Entity Section?

Is JATS overkill? Can there be such a thing as too much documentation?

8.2. Questions about samples

- ◊ What is the most important thing in a tagged sample?
- ◊ Should samples be large (showing context) or as small as possible?
- ◊ Are cookbooks and recipes more useful than prose?
- ◊ How many samples does one element need?

8.3. General Documentation Questions

- ◊ You can't afford both: create a reference manual or a tutorial?
- ◊ What does a user like you need for documentation?
- ◊ How SHOULD Tag Set documentation differ from programming language or scripting language documentation?
- ◊ How do you live without a context table? How many of you do?
- ◊ Whose responsibility is it to document public vocabularies?

8.4. And in conclusion...

- ◊ Is documentation obsolete?
- ◊ How much user documentation can/should be created automatically?
- ◊ Will AI (or smart-whatever) save us from the need for documentation? (smart systems that guide/lead/force users)

9. Other Resources to Support JATS Users

This paper only describes the non-normative Tag Library documentation made available for JATS by the United States National Library of Medicine (NLM), which also sponsors and hosts the DTDs, RNG, XSDs, and all the apparatus of publication and versioning. But there is much more JATS documentation. This section lists but a few of the extensive JATS resources produced by organizations outside of NISO and not part of NLM.

9.1. JATS-List and the Archives of JATS-List

JATS-List [JATS Listserve] is a public email discussion list devoted to JATS and JATS-related topics. JATS users can ask questions, answer the questions of others, or start

discussions on topics of interest. The archives of the JATS-List [JATS List Archives] are publicly available and include all postings to the list since it was started in 2010. (Note: This is an active *friendly* list and infinitely patient with newbies.)

9.2. JATS4R (JATS for Reuse)

JATS4R [JATS4R] is a NISO working group “devoted to optimizing the reusability of scholarly content by developing best-practice recommendations for tagging content in JATS XML.” - JATS4R website

9.2.1. JATS4R Recommendations

JATS4R publishes recommendations on how to use the JATS Tag Set to optimize reuse and interoperability of JATS documents.

Current recommendations include:

Article publication and history dates

How to capture publication dates associated with an article and indicate the version of the article being viewed

Authors and affiliations

How to capture authors and their affiliations in article metadata

Citations (general)

How to capture tagging citations in general

Clinical trials

How to capture clinical trial information in abstracts and other places in an article

Conflict-of-interest (COI) statements

How to capture conflict-of-interest (COI) statements within article metadata

CRediT taxonomy

Updated version of how to tag author contributions using the CRediT taxonomy

Data availability statements

How to capture data availability in a machine-readable section of your JATS XML

Data citations

Updated version of how to tag data citations as references

Display objects (e.g. figures, tables, and boxed text)

How to capture figures, tables and boxes in a standard format in your JATS XML

Ethics statements

How to capture ethics-related content within a specific section and within general content

Funding

How to capture funding information in a specific location so it can be mined for further use, with reference to the Open Funder Registry

General XML recommendation

How to capture general article's XML as a whole

Math

How to capture mathematical content within an article

Peer review materials

How to tag peer review materials such as peer reviews, editor decision letters and author responses

Permissions

How to capture license and copyright information for the article as well as reproduced assets within it

Preprint citations

How to tag preprint citations as references

9.2.2. JATS4R Validation Tool

(From the JATS4R website)

JATS4R produces and makes available an online tool to validate an XML document both against the appropriate JATS DTD (NISO JATS version 1.0, 1.1 or 1.2), as well as against the JATS4R published recommendations.

The source code (MIT-licensed where applicable) is available for the JATS4R user interface, the validator web service, the Schematron rules, and the JATS DTDs.

When a user's XML content is sent to the JATS4R web service for validation, the doctype is logged, for usage metrics, but the rest of the XML document is not stored.

9.3. JATS-Con and the JATS-Con Proceedings

JATS-Con is a (mostly) annual user-level conference [JATS-Con Conference] devoted to JATS and the JATS family of specifications (the three JATS Tag Sets, BITS, NISO STS, etc.). The proceedings of JATS-Con are available in the NLM Bookshelf [JATS-Con] and include as many of the following as are available for each presentation: a formal paper, the presentation slides, and video of the presentation.

9.4. NISO site for comments/suggestions

The NISO website is available for public comment [JATS-Comments]. All comments are reviewed by the JATS Standing Committee and answers posted on this site.

9.5. Local Guidelines

Many organizations and services that ingest JATS documents (including libraries, database services, service providers such as web hosts, and tool vendors) provide guidelines on how they want JATS to be structured for their ingest. Some require particular versions and/or

impose business rules in addition to the validity to the JATS schemas, for example, that all tables and figures must be referenced from within the text.

Guidelines can be quite elaborate. For example, the PubMed Central Tagging Guidelines [PMC Guidelines 2021], includes comprehensive text explanations, tagged examples, and an online style checker that confirms whether an XML file conforms to PMC Style as defined in the PMC XML Tagging Guidelines.

References

- [JATS 2019] American National Standards Institute/National Information Standards Organization (ANSI/NISO). ANSI/NISO Z39.96-2019, JATS: Journal Article Tag Suite, version 1.2. Baltimore: National Information Standards Organization; 2019. Available at: <https://www.niso.org/publications/z3996-2019-jats>
- [JATS Homepage] Journal Article Tag Suite Bethesda (MD): National Center for Biotechnology Information (NCBI), National Library of Medicine (NLM). Available at: <https://jats.nlm.nih.gov/>
- [BITS 2.0] Book Interchange Tag Set: JATS Extension. Bethesda (MD): National Center for Biotechnology Information (NCBI), National Library of Medicine (NLM). Available at: <https://jats.nlm.nih.gov/extensions/bits/>
- [JATS4R] JATS4R (JATS for Reuse). Available at: <https://jats4r.org/>
- [JATS-Con] Journal Article Tag Suite Conference (JATS-Con) Proceedings. Bethesda (MD): National Center for Biotechnology Information (NCBI); 2010-. Available at: <https://www.ncbi.nlm.nih.gov/books/NBK65129/>
- [JATS Listserve] JATS-List — Open Forum on the Journal Article Tag Suite. Available at: <http://www.mulberrytech.com/JATS/JATS-List/>
- [JATS List Archives] JATS-List Archives (jats-list /at/ lists.mulberrytech.com Mailing List Interface). Available at: <https://www.biglist.com/lists/jats-list/archives/>
- [JATS 2021 1.3d2] Journal Archiving and Interchange Tag Library, NISO JATS Version 1.3d2 (ANSI/NISO Z39.96-2019). December 2020 (documentation updated April 2021). Bethesda (MD): National Center for Biotechnology Information (NCBI), National Library of Medicine (NLM). Available at: <https://jats.nlm.nih.gov/archiving/tag-library/1.3d2/index.html>.
- [JATS Home NLM] Journal Article Tag Suite. Bethesda (MD): National Center for Biotechnology Information (NCBI), National Library of Medicine (NLM). Available at: <https://jats.nlm.nih.gov/>
- [STS 2017] NISO Standards Tag Suite (STS) Supporting Materials. (Schemas and Tag Library) Available at: <https://www.niso-sts.org/>
- [JATS-Con Conference] JATS-Con (Journal Article Tag Suite Conference) annual conference. Bethesda (MD): National Center for Biotechnology Information (NCBI), National Library of Medicine (NLM).
- [JATS-Comments] NISO JATS Comment Form. Standardized Markup for Journal Articles: Journal Article Tag Suite (JATS). Baltimore: National Information Standards Organization. Available at: <http://www.niso.org/standards-committees/jats>.

[PMC Guidelines 2021] PubMed Central Tagging Guidelines. Accessed May 11, 2021.
<https://www.ncbi.nlm.nih.gov/pmc/pmcdoc/tagging-guidelines/article/style.html>



2021 The Future of Distributed Markup Systems or ‘Help my package has become too large!’

Karin Bredenberg, Kommunalförbundet Sydarkivera

Jaime Kaminski, Highbury R&D

This paper expands on the themes developed in “Beyond the brick, for the past in the future, you find the archive!” which was presented by the authors at Markup UK 2019. It explores the development of the European Commission’s eArchiving Building Block from the perspective of the underlying XML-based specifications. It will also look at some of the new specifications that have been developed by the Building block since 2019 and considers some key future challenges.

1. Introduction

In 2019 you were introduced to the world of digital archiving and the eArchiving Building Block in a paper titled “Beyond the brick, for the past in the future, you find the archive!” [<https://markupuk.org/2019/webhelp/index.html#ar01.html>]. We explored how common specifications have been used for describing both Information Packages and the Content Information Type Specifications (CITS) that can be placed in an Information Package. This is a new world for some and an old world for others, so let’s recap and introduce you to the challenges that follow.

2. It all started with a declaration

In 2018 Estonia was the chair of the European Union, as the leadership switches every six months on a running schedule, so all countries get to be chair. Estonia is deeply engaged with the digital transformation and has been an inspiration for many.

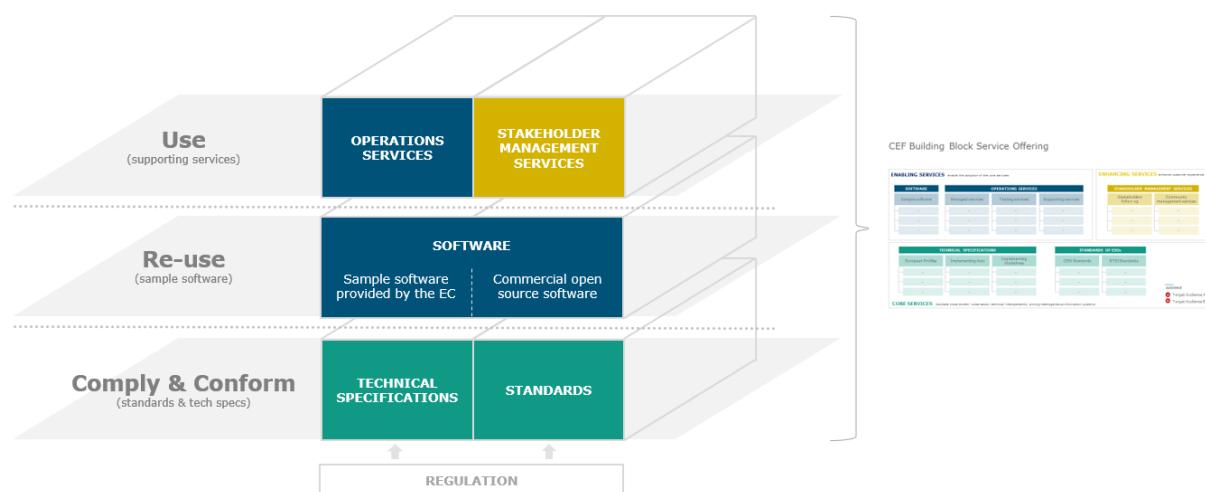
During their presidency, all the European Union Member States and EFTA countries signed the ‘eGovernment Declaration’ in Tallinn on 6 October 2017. The so-called *Tallinn Declaration* [<https://digital-strategy.ec.europa.eu/en/news/ministerial-declaration-egovernment-tallinn-declaration>], with its seven principles, focuses on high quality, user-centric digital public services for citizens as well as seamless cross-border public services for businesses. In December 2020, the Tallinn Declaration was further underpinned by the *Berlin Declaration* [https://ec.europa.eu/isa2/sites/default/files/cdr_20201207_eu2020_berlin_declaration_on_digital_society_and_value-based_digital_government_.pdf], which re-affirms Europe’s deep commitment to fundamental rights and European values and emphasises the importance of digital public services. The Berlin Declaration takes the principles formulated in the Tallinn

Declaration further by enhancing the role of public administrations in driving Europe's digital transformation.

3. The Building Blocks

Let's start with the Connecting Europe Facility *CEF* [<https://ec.europa.eu/inea/en/connecting-europe-facility>] and its *Building Blocks* [<https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/CEF+Digital+Home>]; what are they? The European Union realise that the internet and digital technologies are transforming our world. They also see is that the digital landscape is becoming more diverse, creating challenges for cross-border interoperability and intercommunication. Europe is about working together but, Europeans still face barriers when using (cross-border) online tools and services. The implications are considerable. EU citizens can miss out on goods and services, and businesses in the EU miss out on market potential. At the same time, also the different governments in the EU cannot fully benefit from digital technologies. Therefore, the EU has described the Digital Single Market (*DSM*) [<https://ec.europa.eu/digital-single-market/en>] through which it aims to create a suitable environment for digital networks and services to flourish. The *DSM* is achieved by setting the right regulatory conditions and providing cross-border digital infrastructures and services. So, to support the *DSM*, the Connecting Europe Facility (*CEF*) programme is funding a set of generic and reusable Digital Service Infrastructures (*DSI*), known as Building Blocks. These Building Blocks offer reusable modular capabilities to enable digital public services across borders and sectors. There are currently nine Building Blocks: Big Data Test Infrastructure, the European Blockchain Services Infrastructure, Context Broker, eDelivery, eID, eInvoicing, eSignature eTranslation and eArchiving. The main component of the Building Block is a Core Service Platform, provided and maintained by the European Commission. The Core Service Platform can include technical specifications, sample software and support services depending on the specific Building Block. The *CEF* Building Blocks provide basic capabilities that can be used to facilitate cross-border public service. The foundation of the *CEF* Building Blocks is interoperability agreements between the member states of the European Union. The objective of the Building Blocks is to facilitate interoperability between IT systems so that citizens, businesses and administrations can benefit from seamless digital public services wherever they may be in Europe.

Figure 1. The building block layers.



For each Building Block, the European Commission provides a Core Service Platform that consists of three layers:

- ◊ At the core of each building block is a layer of standards and technical specifications;
- ◊ for certain Building Blocks, a layer of compliant sample software exists to facilitate the implementation of the technical specifications and standards;
- ◊ A layer of services (e.g. conformance testing, help desks, onboarding services, etc.) enables the adoption of the technical specifications and standards meant for use (which varies depending on the Building Block).

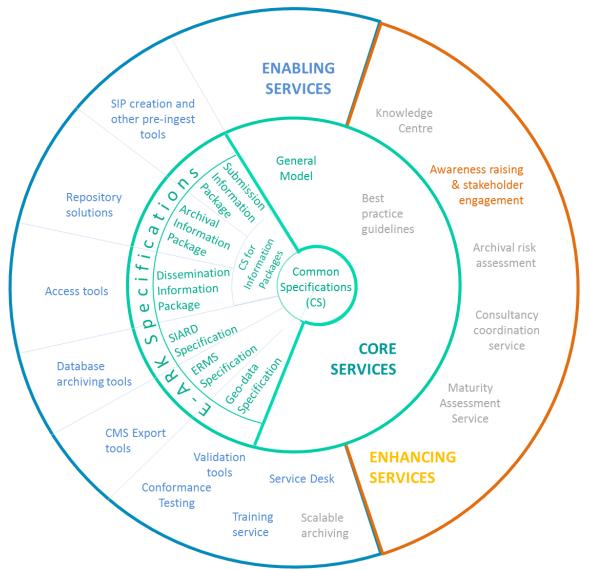
All this means that the Building Blocks can be combined and used in projects in any domain or sector at the European, national or local level.

Can it be as simple as just described with the Building Blocks? No, during 2021, CEF is ending and will be transitioned to the Digital Europe Program (*DEP*) [<https://eufundingoverview.be/funding/digital-europe-programme>], which will be funded for seven years. This brings challenges to all the Building Blocks. At the writing of this paper, the EU member states have still not finalised the details of the DEP programme. The programme itself will be housed in the eHealth part of the digital work within the EU nothing more is known.

4. Recap of the eArchiving Building Block

The *eArchiving Building Block* [<https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eArchiving>] supported by The Connecting Europe Facility (*CEF*) [<https://ec.europa.eu/inea/en/connecting-europe-facility>] and the European Commission (*EC*) [<https://ec.europa.eu>] creates common specifications based on pre-existing standards that everyone can use to transfer data. eArchiving aims to provide the core specifications, software, training and knowledge to help data creators, software developers, and digital archives tackle the challenge of short, medium and long-term data management and reuse in a sustainable, authentic, cost-efficient, manageable and interoperable way. The birth took place in 2018 and was a long one ending in 2019. The result was several specifications for the different types of information packages found in the *OAIS Reference model* [<https://www.iso.org/standard/57284.html>] and Content Information Type Specifications (*CITS*) for structuring the content to be placed in the package. Tools were also developed, but let's focus on the foundation, the specifications. In November 2019, a new two-year project named E-ARK3 was started, and is developing more specifications and enhancing those created previously. But where does this winding path take us? To more specifications and more data to transfer? The path will become more and more convoluted the longer we get on the journey of describing data as content in a content information type specification.

Figure 2. The eArchiving building block and its services and specifications.



5. Standards, de facto standards, and specifications

Jenn Riley's *Visualisation of the Metadata Universe* [<http://jennriley.com/metadatauniverse/>] has been around for a while but still gives the best overview of standards used in the cultural sector, archives, libraries and museums. In the image, numerous standards are displayed in the context of their function and where they are used.

Figure 3. The visualization of the Metadata Universe by Jenn Riley. (The recommendation is to look on-line)



Almost all standards on the metadata map created by Jenn Riley have an XML format available described with a DTD or an XML schema. For the XML schemas, both the ISO standard RelaxNG as well as W3C XML-schema formats are used. The choice depends solely on the skills of the creator of the schema. At the same time, it is also common to ensure that different types of schemas are available, so transformations from RelaxNG to XML schema and vice versa are often used. DTD is still around because old software is still in use, and it is often based upon using a DTD.

The most common way to use the different standards is to write a specification that describes a profile for your use case of the standard, which then is implemented in the setting you are operating. But here comes the part that the eArchiving building Block is tackling; we all write our own specifications on how to use the standards, so no one understands what anyone else has done. We need a set of common developed

specifications that can be used and publicised in one central location, so they are easy to find for all. That is what eArchiving is trying to achieve.

6. The eArchiving specifications and its different faces

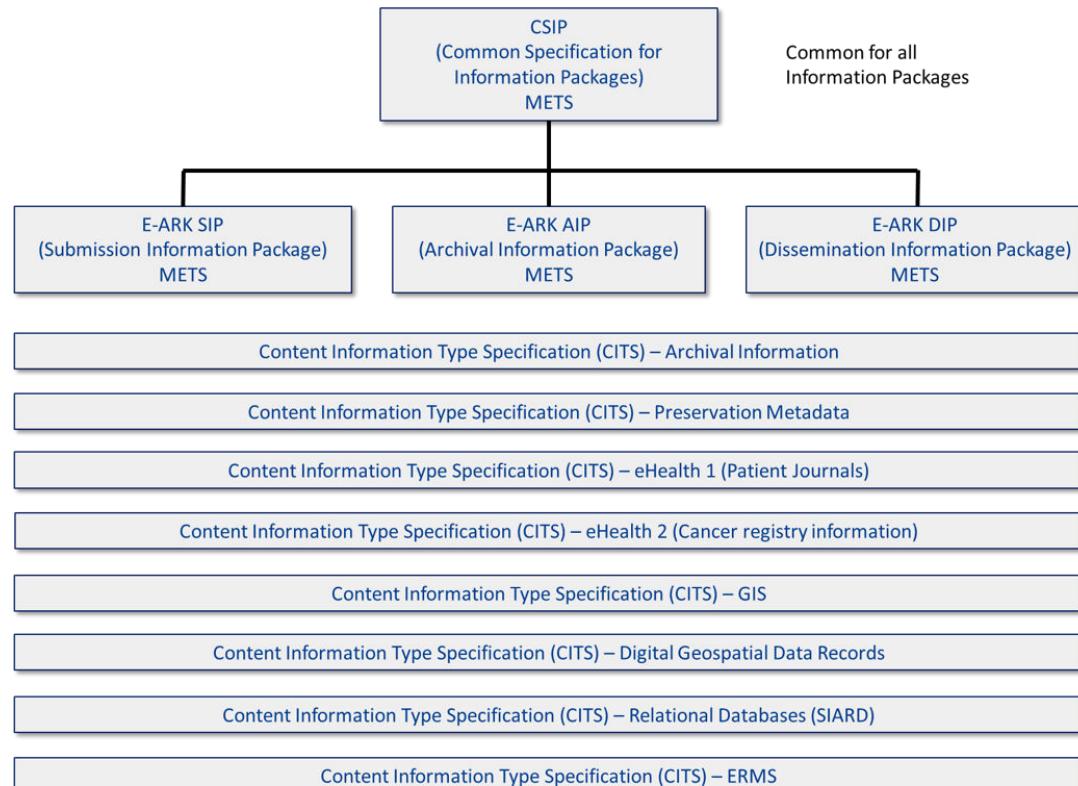
The core of eArchiving is the Information Package specifications. These describe a common format for storing bulk data and metadata in a platform-independent manner, authentic and understandable over the long term. The specifications are ideal for:

- ◊ migrating valuable long-term data between different generations of information systems,
- ◊ transferring data to dedicated long-term repositories (such as digital archives), or
- ◊ preserving and reusing data overextended (and shorter) periods and different generations of software systems.

The layer outside the core is the Content information Type Specifications (CITS). These describe the content itself and ensure that the standards are suitable for transferring the content in a unified way.

The CITS are the specifications that currently are increasing in numbers. The information package specification is undergoing refinement and testing within the project by the team that is creating a validation tool for the information package. A first version of the validation has already been released, but it will be extended with more tests to ensure an information package follows the main specification for information packages.

Figure 4. The eArchiving building block and the different specifications building up a Information Package (2021)



With these two different types of specifications (all currently based on XML), an information package filled with content can be described and transferred. XML has long been the go-to format for the long-term preservation of information, mainly because it can structure

information in a understandable way to both humans and machines. There have been concerns and comments raised regarding Json taking over the role of XML. However, in the archival setting, even if JSON is easier to use as a programmer in the long term, XML wins because of its readable elements and attributes explaining the information or what we call content placed in the document. However, at the same time, XML needs to be used wisely with the correct element and attribute names to facilitate understanding of the human reading of the XML document both now and in the future. There are also other formats used like tiff and pdf, but XML is the format to go to in most cases to reuse the information.

7. Defining a CITS with the help of experts

The information package itself is when you have the knowledge easy to create. But when we talk about the content, the challenges grow. For the content, the different content experts need to support creating the specification for the specification. A group of experts is not always easy to find since they need to know the content. They also need to understand the concept of specifications being able to use in different settings and by different users. For now, the creators of the specifications are part of the project. In the future, endorsement of specifications already in use and connections with content experts creating new specifications will be crucial to extend the numbers of specifications to cover all the content that can be found.

At the same time, the CITS that have been developed and are in development have shown the need for different ways of creating a CITS. The different ways also give different possibilities in validation services possible to create. Not to forget, how do we get this content out of a proprietary system?

The different ways are easiest described:

- ◊ The specification endorses the use of an already created and widespread specification and provides minor detail on how it is placed into an information package.
- ◊ The specification focuses on the placement of content in an information package. Some ordering of the content to make it understandable in the long term is described.
- ◊ The specification gives thorough information on how the content itself is structured and how its content and files are placed in the information package.

This means that the challenge of defining a new CITS needs to not only be able to find the real experts on the content it is also about how many of the requirements will be placed in the specification where a different type of expert is needed. The number of requirements needed to be specified usually solves itself based upon the type of content and the prior work describing it. At the same time, an endorsement will be an important mechanism moving forward. An endorsement will include a description about how to place the content in the specification and then point to the content specification being maintained by an expert group on the content. An example here is the CITS for Archival Information which only gives a list of possible standards to use. The use of the archival standards is described at an expert group managing profiles for this.

8. A database specification

Databases are crucial for long-term preservation. In some cases, there will be a specification suitable for mapping from the database to an XML schema and thus giving an XML document (or more) which makes it possible to easily understand the content in the future. An example of this is Record Management Systems that can be mapped towards the CITS for Electronic Records Management Systems (CITS ERMS). But this is not always the case. There will always be databases that need to remain databases. You might think this is an easy task; just save the database dump, and we can just restore it

ten years later and continue as before, but I can assure you it is not that easy. Databases need to be transformed into a sustainable format to ensure they can be preserved long term. The eArchiving Building Block uses the SIARD standard (Software Independent Archiving of Relational Database) developed by the Swiss Federal Archives (SFA) [<https://www.bar.admin.ch/bar/en/home.html>] and now maintained by SFA and the *DILCIS Board* [<https://dilcis.eu/>]. With the help of the available tools, SIARD transforms the database into an XML format. But many other files can be hidden in a database in the form of BLOBs and or CLOBs, which means that the total size of a database can be huge, and the number of files not suitable for transforming to XML enormous. So how do you transform a database with data in from simple values to an XML format with the files extracted and referenced in the XML document? In the transformation, the database content itself becomes XML, and the BLOBs and CLOBs become files referenced in the XML document.

At the same time, we want to transfer this XML document with its files to an archive, which means we want to put it into an information package to include some surrounding information, such as example definitions and explanations of value lists. An information package also adds the possibility of controlling checksums, to confirm that what has been transferred is what has been received. This means that all files are referenced in two XML documents. First, in the XML documents produced by the transformation, and second, in an information package XML document.

The complexity grows. Several questions arise:

- ◊ How big can an information package become?
- ◊ Can it grow indefinitely?
- ◊ Can we put all the files in one folder and describe them in just the two XML documents?

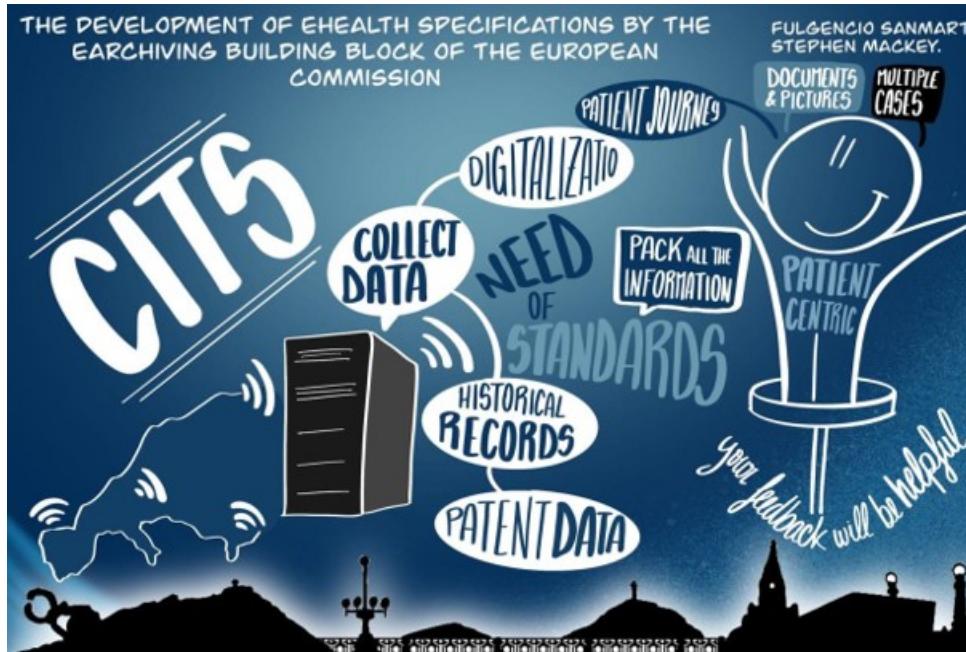
Going through the questions, we can see that all XML documents and files can be in one package, but that might give us an XML document that takes 24 hours to validate if the database was filled with files. Then time is needed for checking all the checksums needed to ensure that what was supposed to be transferred arrived without losses or spurious additions.

This means that we need to set up recommendations and rules about splitting the package into more packages, so the information is divided into more than one information package. This is where we are today, making the best recommendations on how to split a huge amount of content into different information packages.

9. Other “big” information types

Databases are not unique when it comes to creating challenges regarding the number of bytes that can be hosted or their content. Another area is eHealth. eHealth is an area where we can see many different systems being used. These include proprietary and bespoke systems alongside different standards focusing on many elements of healthcare provision. The central element for eHealth is clearly defined; it is the patient and the patient's journal. This has its own challenge since usually, the focus for an archival transfer is the content. During this project, a specification has been developed based upon work performed in Norway for handling the transfer of deceased patient's journals to the Norwegian Health Archive. The specification, in this case, does not go into which files are present; instead, it focuses on creating an information package with the patient-centric focus, which means the content is as it is just placed in the correct place. But considering the content with maybe numerous MRI scans and so on also, it needs to be possible to split the content into more than one information package to facilitate the transfer. A small hint – in this case, the splitting will follow the same rules defined for the databases.

Figure 5. Marisa Merino Hernandes illustrated the eHealth1 specification in the following way.



This eHealth specification will not be the only health specification, but it is the one focused on the patient. Another health specification that has been developed in the project relates to cancer registries and their transmission of information to different cancer information aggregators. For this specification, the focus has been on placing the content into the package with the addition of the extra information needed when you transfer the information for long-term preservation.

10. How do we facilitate oversized information packages?

There will always be a problem with the package being too large to handle. The solution is not to get more powerful computers to handle the packages. It needs to be possible to achieve in different hardware and software environments and with varying staff skills. We cannot count on having access to the funding to get the hardware we want, but we still have to preserve the now for the future and so need to describe how we can split an information package into more parts. And not to be forgotten, we need to rebuild it again in the end when the content is going to be used. Moving forward will see more emphasis on the use of distributed systems, but not in the sense you most likely are thinking. Here we are more in the area of spreading the placement of content to different locations with the information package pointing to these locations instead of everything being contained in the information package.

11. Where do I find my packages?

A question that we hope you appreciate is not easy to answer, but the work is ongoing to facilitate distributed content to keep the size of an information package manageable no matter whether it is distributed into interlinked packages or distributed storage with a central information package description.



An improved diff3 format using XML: diff3x

Robin La Fontaine, DeltaXML

Nigel Whitaker, DeltaXML

There is no doubt that the diff and diff3 format has established itself as a well-used de-facto standard. It might seem presumptuous to suggest that it could be improved, or indeed that it needs to be improved. However, the original premise of line-based text files as the subject matter is now out of date with more structured information being the norm. Often this is in the form of programming source code where the layout tends to remain fairly consistent through edit cycles, but increasing use of JSON and XML pose particular difficulties for the simple line-based structure of diff3.

In our paper at MarkupUK in 2019, we discussed some of the issues and suggested some minor improvements to diff3. These changes suffered from the common complaints of a retro-fit in that they did not sit comfortably with the original and only did half a job. The prevalence of GUIs also suggest that the actual syntax of a diff file is not as important as it was in that the emphasis has changed from human readability to interchange between two applications, for example between a git 'merge driver' and a git 'mergetool'. For these reasons it seemed better to consider a different approach using the tools and formats that are now in common use, for example XML or JSON.

What might diff3 look like as an XML format? Would the advantages of a new format make it worth swapping from the tried and tested diff3? Could existing GUI software easily adapt to a new format and, perhaps, even be simpler as a result?

1. Introduction and Background

This paper is a sequel to "An Improved diff3 Format for Changes and Conflicts in Tree Structures" [1] and again is focused on the diff3 format rather than the diff3 executable application. In this paper we will develop an XML alternative to the diff3 format from GNU diffutils [2]. There are many possible outputs from diff3 but the one we are interested in is the one that provides a merged file result with conflicts marked up, i.e. the '-m' option on the command line.

Many users do not view diff3 data directly or invoke diff3 itself, instead it is often invoked by a version control systems such as git or mercurial when the users merge a branch, graft or cherry-pick, rebase or change branches with working directory changes.

A characteristic that we seek from the start is that as the number of changes tends to zero, so the diff file tends to resemble the original files. This is desirable in that minimal processing is needed for few changes and human understanding is improved simply because when the diff and the original files are very similar they will look very similar.

We distinguish between the 'carrier' syntax which is the diff3 alternative, and the 'payload' which is the content of the file(s) being compared or merged.

In our previous paper, we considered both XML and JSON as a candidate for the carrier syntax and established that XML is a more natural fit, with the payload of the original files being, typically, not XML. However, of course this could be applied to XML itself and then there is likely to be at a human readability level a confusion between the payload and the carrier, and we will look at that in this paper.

2. Developing an XML syntax for diff3x

First we look at the trivial but important example of a diff3 for three identical files, i.e. there are no changes. As we are using XML there is a minimum overhead of the start and end tags, so if the example files are all the same, as shown below:

Table 1. Three identical files

A.txt	O.txt	B.txt
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6

The XML representation would be as shown below and note that white space needs to be preserved, or CDATA could be used.

```
<diff3x>1
2
3
4
5
6</diff3x>
```

Now we can move on to represent actual changes.

We need an example to show the syntax and we will use the same example as in the previous paper because it also allows us to illustrate how the XML syntax can potentially represent a richer view of the differences. For clarity we repeat the example here. The example is based on this paper, "A Formal Investigation of Diff3" [3]. The example consists of three text files with numbers on each line, the files are denoted A.txt, B.txt and the 'old' file O.txt as shown below:

Table 2. Mismatched sequences

A.txt	O.txt	B.txt
1	1	1
4	2	2
5	3	4
2	4	5
3	5	3
6	6	6

The way these are combined into the two diffs, A+O and O+B are shown in the table below.

Diff3 alignment across two diffs

A	O		O	B		A	O	B
1	1		1	1		1	1	1
4			2	2		4		
5			3			5		
2	2		4	4		2	2	2
3	3		5	5		3	3	
4			3			4	4	
5			6	6		5	5	
6	6					6	6	6

The last three columns show how the two diffs are combined. Note that the yellow match shows where all three files align - and this is important because it is the data between these alignment points that are considered as units of change. Now we can look at the diff3 output using the -m option:

```

1
4
5
2
<<<<< A.txt
3
||||||| 0.txt
3
4
5
=====
4
5
3
>>>>> B.txt
6

```

How might this look as XML? Because we are only looking at a maximum of three files it seems reasonable to have a specific element to represent each one. However, we also need to record the original file names and these could be shown as attributes on the root element. So, the above might be represented in XML as follows.

```

<diff3x a="A.txt" b="B.txt" o="O.txt">1
4
5
2<choice3><a>
3</a><o>
3
4
5</o><b>
4
5
3</b></choice3>
6</diff3x>

```

The element <choice3> introduces a three-way choice between the three original files.

2.1. Showing Non-conflicting Changes

Now that we have moved into an XML syntax world, we have the opportunity for a richer representation to show non-conflicting changes. It is often useful to see these to understand how a merge has been handled and provides more context when resolving conflicts. In this example we could show the origin of the '4 5' data that was added by A.

To show this we need to have elements defined for the limited combinations of the three input files. We can do this as `<ao>` for data in both O and A, and similarly `<ab>`, `<ob>`. We do not need `<aob>` because any lines matched across all three will be present as a child of the root element. Any non-conflicting change will have one of these dual origin elements as one of the two options. As these are non-conflicting changes, they are a resolved choice and we can indicate this with an attribute to show which option has been included. This attribute could also be used to indicate which option or options in a conflicting choice is to be included in the final result. A GUI might provide a user with the ability to undo this choice.

```
<diff3x a="A.txt" b="B.txt" o="O.txt">1<choice2><a include="true">
4
5</a><ob/></choice2>
2<choice3><a>
3</a><o>
3
4
5</o><b>
4
5
3</b></choice3>
6</diff3x>
```

2.2. Showing linked changes to JSON Structure in diff3x

For JSON, the issue of handling curly braces (for objects) and square brackets (for arrays) is similar to XML start and end tags. Again, some representation of connected change is needed to maintain syntactic correctness.

Object members and array members are comma separated and this is tricky to get right in some situations. The syntax is shown below.

```
object = begin-object [ member *( value-separator member ) ]
                  end-object
array = begin-array [ value *( value-separator value ) ] end-array
```

These are the six structural characters:

```
begin-array = ws %x5B ws ; [ left square bracket
begin-object = ws %x7B ws ; { left curly bracket
end-array = ws %x5D ws ; ] right square bracket
end-object = ws %x7D ws ; } right curly bracket
name-separator = ws %x3A ws ; : colon
value-separator = ws %x2C ws ; , comma
```

Insignificant whitespace is allowed before or after any of the six structural characters.

```
ws = *(
      %x20 / ; Space
      %x09 / ; Horizontal tab
```

```
%x0A / ; Line feed or New line
%x0D ) ; Carriage return
```

Here is an example of a change to an array of strings.

Table 3. JSON structural change

A.txt	O.txt	B.txt
[[12, 13, 14], 20, 21, 22]	[12, 13, 14, 20, 21, 22]	[[12, 13, 14, 20, 21, 22]]

This could be represented in the diff3 format as shown below, but note that this is not how diff3 would process the above files which each have a single line. However the result below could be generated by careful use of line breaks in the input files.

```
[
<<<<<< A.txt
[
||||||| O.txt
=====

>>>>> B.txt

<<<<<< A.txt
||||||| O.txt
=====

[
>>>>> B.txt
12,13,14
<<<<<< A.txt
]
||||||| O.txt
=====

>>>>> B.txt
,20,21,22
<<<<<< A.txt

||||||| O.txt
=====

]
>>>>> B.txt
]
```

The above will only produce syntactically correct results if the correct choices are made, which is not easy. It is also not easy to see what is going on in the diff3 file because of the extra whitespace lines that have to be inserted to make this work.

We introduce here a way to connect the relevant consistent choices so that if the '[' is selected then the appropriate choice of the end ']' is also made automatically.

A choice consists of one or more options. An option may have an id. An option may also have a select attribute which provides a boolean value made up of one or a combination

of other ids and if these are 'true', i.e. have been chosen to be included, then this option is selected automatically.

This could be represented in diff3x as shown below. We show here the id attribute on an option to identify it uniquely within the file. We then use that id to reference that option where another option needs to be linked to it. In this case, the '[' in the A.txt has `id="a42"` and this is then referenced in the select attribute of the corresponding ']' option later in the file. Similarly the pair of square brackets in B.txt are linked with the id "b44".

```
<diff3x a="A.txt" b="B.txt" o="O.txt">
[
<choice2>
<a id="a42">[</a>
<ob/></choice2>
<choice2>
<b id="b44">[</b>
<ao/></choice2>
12,13,14
<choice2>
<a select="a42">]</a>
<ob/></choice2>
,20,21,22
<choice2>
<b select="b44">]</b>
<ao/></choice2>
]</diff3x>
```

Note that the '[' in A could be selected either instead of or as well as the ']' in B even though they are at the same position in the array. Whichever choice is made, the other choices with a select attribute identifying the same id are chosen and the result is syntactically correct. This is a powerful way to represent connected choices.

2.3. Representing JSON Separator Change in diff3x

The problem with separators is that they cannot consistently be associated with either the start or the end of each item (member for object and value for array) because if there is only one item then no separator is needed. Therefore maintaining correct syntax when items are added or deleted is not trivial. As mentioned above, the diff3 format does not allow consecutive choices without 'anchor' data between, so it is necessary to group consecutive items that may be added or deleted into one choice. This apparent restriction does lead to a greater likelihood of the syntax of each choice being consistent.

Here is an example of a change to an array of strings.

Table 4. JSON array value change

A.txt	O.txt	B.txt
["one", "two"]	["one"]	["three", "four"]

This could be represented as shown below. Note here that we are not showing the result of running 'diff3 -m' but rather we have run an XML aware comparison so we have results that we want to express in the diff3 format.

```
[<<<<< A.txt
"one", "two"
```

```
||||||| 0.txt
"one"
=====
"three", "four"
>>>>> B.txt
]
```

The above will produce syntactically correct results though it is not ideal because it would be more natural to choose the values separately rather than as a complete list. This can be achieved with diff3x as shown below. Here we have been able to perform the merge automatically, there are no conflicts but clearly it is useful for the user to see these and have the option to undo or change a choice.

We introduce here a third type of choice, the <autoInclude>. This is a choice that we do not expect the user to review but rather it is a single option that is chosen automatically based on one or more other selections. In this case, the commas are inserted only when they are needed.

```
<diff3x a="A.txt" b="B.txt" o="0.txt">
[
<choice2>
  <ao id="ao42">"one"</ao>
  <b id="b44" include="true"></b></choice2>
<autoInclude select="AND(ao42 a52)">, </autoInclude>
<choice2>
  <a id="a52" include="true">"two"</a>
  <ob id="ob53"></ob></choice2>
<autoInclude select="AND(OR(ao42 a52) b62)">, </autoInclude>
<choice2>
  <b id="b62" include="true">"three", "four"</b>
  <ao id="ao63"></ao></choice2>
]</diff3>

Result of this choice is:
["two", "three", "four"]
```

This is more complex in getting the logic correct for insertion of commas. XML users will be pleased that XML attributes are not comma separated!

3. Preserving well-formed tree structure in diff3x

In this section we explore the issues of preserving the well-formed structure of XML when presenting choices in diff3x.

3.1. Representing XML Element Tag Change in diff3x

XML tags present a problem for diff3 format in that it is in general not possible to ensure a well-formed result without unacceptable duplication of content. To handle tag changes in diff3x we need to treat the XML payload, i.e. the XML that is the subject of change, as text and so escape it with CDATA markers. Later in this paper we look at treating an XML payload as XML, which is more natural but as we shall see, it is not possible to represent tag changes in that approach.

Here is an example of a change of structure.

Table 5. XML tag change

A.txt	O.txt	B.txt
<p><p>This is a long paragraph where most of it has been made either bold or italic, but the rest of the paragraph remains unchanged - there is no change to the text so we do not want it duplicated.</p> <p></p></p>	<p><p>This is a long paragraph where most of it has been made either bold or italic, but the rest of the paragraph remains unchanged - there is no change to the text so we do not want it duplicated.</p> <p></p></p>	<p><p>This is a long paragraph where most of it has been made either bold or italic, but the rest of the paragraph remains unchanged - there is no change to the text so we do not want it duplicated</italic>.</p> <p></p></p>

This could be represented as shown below, but there is duplication of unchanged text which is confusing because if there had been a small change the user would have found it difficult to see. Note that in this example the payload is XML but this is not seen as part of the XML of the carrier, the payload is treated as text because it is enclosed in the CDATA sections.

```

<diff3x a="A.txt" b="B.txt" o="O.txt"><![CDATA[<p>This is a long paragraph where ]]>
<choice3>
<a><![CDATA[<strong>most of it has
been made either bold or
italic, but the rest of
the paragraph remains
unchanged - there is no
change to the text so
we do not want it
duplicated</strong>
]]></a>
<o><![CDATA[most of it has
been made either bold or
italic, but the rest of
the paragraph remains
unchanged - there is no
change to the text so
we do not want it
duplicated]]></o>
<b><![CDATA[<italic>most of it has
been made either bold or
italic, but the rest of
the paragraph remains
unchanged - there is no
change to the text so
we do not want it
duplicated</italic>
]]></b></choice3>
<![CDATA[. </p>]]>
</diff3x>

```

We can improve this significantly and use the id attributes as described above to ensure consistent choices. In this case we have given the options within the two connected choices an id value. Thus if option <a> is selected in one choice with id="a42" then the choice

with id="a420" is automatically selected. We have also added 'vice versa' attributes so that the same would happen the other way round: if the end tag was selected then the corresponding start tag would also be selected.

```
<diff3x a="A.txt" b="B.txt" o="O.txt"><![CDATA[<p>This is a long paragraph
where ]]>
<choice3>
<a id="a42" select="a420" include="true"><![CDATA[<strong>]]></a>
<o/>
<b id="b43" select="b430"><![CDATA[<italic>]]></b></choice3>
most of it has
been made either bold or
italic, but the rest of
the paragraph remains
unchanged - there is no
change to the text so
we do not want it
duplicated
<choice3>
<b select="b43" id="b430"><![CDATA[</italic>]]></b>
<o/>
<a select="a42" id="a420"><![CDATA[</strong>]]></a></choice3>
<![CDATA[. </p>]]>
</diff3x>
```

If the GUI tool allowed multiple options to be selected, then both `` and `<italic>` could be selected - provided of course they appeared in the correct nested order. We have assumed that the user would need to have the knowledge to know if the selection of two options was appropriate.

3.2. Representing XML Attribute Change in diff3x

XML attributes present a particular challenge for diff3 format, and this can be improved using an XML representation.

It is worth noting an issue with the diff3 format in this situation, because it is quite subtle but one that we can resolve in diff3x. The lines in the input files at which all three files align (the lines are all equal) are considered 'anchor' points. All of the lines between anchor points is considered to be a single choice - and when there is some kind of conflict then the choice is left for the user to select. The interesting consequence of this structure is that it is not possible to have two consecutive choices without a separator which requires a commonality between all three files, i.e. an anchor point. For structured data it would be natural, for example, to provide choices about attributes in a manner that allows each attribute to be chosen separately, but the diff3 format dictates that two adjacent changes are seen as one choice.

We can improve this with diff3x because we can easily have adjacent choices without any ambiguity.

Table 6. XML Attribute value change

A.txt	O.txt	B.txt
		

These changes can be represented in diff3x as shown below.

```
<diff3x a="A.txt" b="B.txt" o="O.txt">
<![CDATA[<span id="23">]
```

```
<choice3>
<a>class="two"</a>
<o>class="one"</o>
<b>class="three"</b></choice3>

<choice3>
<a>dir="rtr"</a>
<o>dir="TBA"</o>
<b>dir="ltr"</b></choice3>
<![CDATA[</span>]]></diff3x>
```

There is another representation that takes the common attribute name out of the choice but this may be less easy for a user to see what is happening by inspection of the diff3x but a GUI could provide an intuitive interface. For illustration, in the first class attribute we have included the attribute value quotation marks in the options, and for the dir attribute we have put them outside - this is a matter of choice, either approach can be represented by diff3x.

```
<diff3x a="A.txt" b="B.txt" o="O.txt">
<![CDATA[<span
class=]]><choice3><a>"two"</a>
<o>"one"</o>
<b>"three"</b></choice3>
dir=<choice3><a>rtr</a>
<o>TBA</o>
<b>ltr</b></choice3>"<![CDATA[</span>]]></diff3x>
```

4. Nested Changes

As diff3x is an XML representation we here consider going one step further and making use of the fact that the representation is hierarchical to support hierarchical or 'nested' change. A nested change is a change in one branch that modifies something that has been removed in another branch.

We will look at an XML example, showing nested changes.

Table 7. XML nested data example

A.xml	O.xml	B.xml
<author> <personname> <first>Nigel</first> <last>Whitaker</last> </personname> <address> <ph>01684 532141</ph> <st>Geraldine Rd</st> <city>Malvern</city> <country>UK</country> <post>WR14 3SZ</post> </address> </author>	<author> <personname> <first>Nigel</first> <last>Whitaker</last> </personname> <address> <st>Geraldine Rd</st> <city>Malvern</city> <country>UK</country> <post>WR14 3SZ</post> </address> </author>	<author> <personname> <first>Nigel</first> <last>Whitaker</last> </personname>

In the above example one branch, B.xml, has deleted the address sub-tree which the other branch has modified with an added phone number. As we are now using an XML representation that has a tree structure we can also make use of this structure in the result.

In order to distinguish between a simple <ao> option and a nested choice, we introduce the <aoChoice> element. This is a mixed content element that can include <a> or <o> elements as well as text.

```
<d:diff3x a="A.xml" b="B.xml" o="O.xml"><! [CDATA[
<author>
<personname>
  <first>Nigel</first>
  <last>Whitaker</last>
</personname>]]>
<d:choice3>
  <d:aoChoice include="true"><! [CDATA[
    <address>]]>
    <d:a><! [CDATA[
      <ph>+44 1684 532141</ph>]]>
    </d:a>
    <! [CDATA[
      <st>Geraldine Rd</st>
      <city>Malvern</city>
      <country>UK</country>
      <post>WR14 3SZ</post>
    </address>]]>
  </d:aoChoice>
  <d:bChoice/>
  </d:choice3><! [CDATA[
</author>]]>
</d:diff3x>
```

Here we can see that by allowing nested change we can avoid the repetition and make it easier for a human to understand. However, in order to support this in the format we have to introduce quite a number of new elements and different rules. This adds to the complexity of the format and moves us a lot further away from the original elegant simplicity of diff3. Keeping diff3x as simple as possible seems preferable to including the ability to represent a subset of changes as nested changes.

For this reason we have elected not to include the ability to represent nested changes in diff3x.

5. Should an XML Payload be represented as XML or text?

If our payload is XML, and our carrier is XML, does it make sense to embed the payload as XML rather than text? At first sight this seems to be an attractive proposition, and an example is shown below. For clarity we have put the diff3x elements into their own namespace, using a d: prefix.

Table 8. XML data example

A.xml	O.xml	B.xml
<author>	<author>	<author>
<personname>	<personname>	<personname>
<first>Nigel</first>	<first>Nigel</first>	<first>Nigel</first>
<last>Whitaker</last>	<last>Whitaker</last>	<last>Whitaker</last>
</personname>	</personname>	</personname>
<address>	<address>	<address>
<st>Geraldine Rd</st>	<st>Geraldine Rd</st>	<st>Geraldine Rd</st>
<city>Malvern</city>	<city>Malvern</city>	<city>Malvern</city>

A.xml	O.xml	B.xml
<country>UK</country> </address> </author>	<country>UK</country> <post>WR14 3SZ</post> </address> </author>	<country>UK</country> <post>PQ9 5XY</post> </address> </author>

This could be embedded in the XML diff3x carrier as shown below.

```
<d:diff3x a="A.xml" b="B.xml" o="O.xml">
<author>
<personname>
  <first>Nigel</first>
  <last>Whitaker</last>
</personname>
<address>
  <st>Geraldine Rd</st>
  <city>Malvern</city>
  <country>UK</country>
<d:choice3>
  <d:a/>
  <d:o>
    <post>WR14 3SZ</post>
  </d:o>
  <d:b>
    <post>PQ9 5XY</post>
  </d:b>
</d:choice3>
</address>
</author>
</d:diff3x>
```

This looks good and useful, but it has limitations. The main one is that it would not be possible to show changes to attributes in situ, even if we resorted to XML processing instructions. Other approaches to this include representing attribute change by introducing an XML element immediately following the end of the start tag.

Also, it would not be possible to show changes to element tags, e.g. if a `<bold>` was changed to `<italic>`.

These limitations could be overcome by making the diff3x more complex but there is no good reason to do this because it takes away the primary purpose of the diff3x carrier, which is to represent changes to a text file. As an XML payload can be considered to be a text file we should treat it as such to be consistent with the handling of JSON or any other payload.

6. Saving Selected Options

Some choices are for a user to decide which option(s) to include, others can be included automatically either as non-conflicting changes or as a direct result of other selections. If we are able to indicate in the format which options are currently selected, then we can save the state of the document during the resolution process. In addition, this would allow a user to change which option is selected in a choice. It would be advantageous for a GUI to allow not only a single option to be selected but two or more options - for example if A and B have added something different at the same position it may be correct to select neither, one or both options.

7. Comparing diff3 format with diff3x

Although human readability is not the prime concern, it is worth looking at how the files might appear. The table below shows the example first used in Table 2 [109] as diff3 and the corresponding file in diff3x. We have not included the CDATA markers in XML here, but have added an `xml:space="preserve"` attribute indirectly via the schema.

Table 9. diff3 and diff3x representations

diff3	diff3x (XML)
<pre> 1 4 5 2 <<<<< A.txt 3 0.txt 3 4 5 ===== 4 5 3 >>>>> B.txt 6 </pre>	<pre> <diff3x a="A.txt" b="B.txt" o="0.txt">1 4 5 2<choice3><a> 3<o> 3 4 5</o> 4 5 3</choice3> 6</diff3x> </pre>

It is worth noting that another advantage of the diff3x representation is that choices within a line can be represented quite naturally, so this overcomes the significant limitation of diff3 which is only able to show changes to a complete line.

Table 10. Characteristics of diff3 and diff3x

Characteristic	diff3	diff3x	Comment
No processing needed for unchanged file	***	**	
Preserve line structure	***	***	XML needs CDATA
Good for text editor (by hand)	**	*	
Intelligent changes	**	***	
Nested changes	*	***	
Changes within a line	*	***	
Show all resolved merges	*	***	
Show changes to JSON data	**	***	
Show changes to XML data	*	***	
Process the changes in the format of the native payload	-	*	Important to note that this is really not possible - a delta format in the native format is needed.

The table does show significant advantages of having an XML representation of diff3 when communicating merge results from a merge driver to a GUI mergetool.

8. Future work

In previous conferences and events we have shown and discussed different approaches to resolving and processing merge results using XML based technologies. We have shown a conflict resolver using an OxygenXML plugin and one based on Schematron Quick Fixes (SQF) 4. These work well with an XML payload, but aren't directly usable with other forms of structured content such as JSON. At DeltaXML we are finding that customers want to evaluate our tools in the browser and fewer users are willing to download and install a ZIP package with our code/APIs. We have tools/services for XML and JSON and display comparison and merge results in the browser, but we do not provide an interactive merge resolver. Using diff3x would help to provide this capability for both XML, JSON and potentially other structured formats.

In 5 we discussed what we feel are inefficiencies/shortcomings in many git workflows where a mergetool would repeat the work done in the merge driver, often inconsistently. The format presented here is one possible solution that could be used to address these issues. This would allow effort spent understanding structured merged techniques to be concentrated/shared and not repeated in different merge tools. We would welcome feedback from the wider git user community and also from those interested in mergetools.

9. Conclusions

In this paper we have introduced an XML update to diff3 which we have called diff3x. With diff3x, we can integrate more modern structure-aware comparison tools and represent choices that are consistent with the structure and more likely to provide well-formed or valid results.

We have considered the representation of nested changes but concluded that the additional complexity to the diff3x format is unlikely to be worth the small gain.

The diff3x format can take plain text as its payload. If the payload is XML it would need to be escaped for example with CDATA and treated as text. We considered whether, if the payload is also XML, this could be treated as XML and although this is possible we concluded that treating an XML payload as text has the advantage that tag changes and attribute changes can be represented directly in situ. Our experience with XML tools and technologies has allowed us to develop XML-centric resolvers, but these have the limitation that they do not work with JSON and other non-XML formats. They still have benefits and their applications, for example when considering nested change or n-way merge. As such they complement the benefits of diff3x.

In a related paper [5] we identified some issues in version control systems that caused inconsistency and confusion to users. One solution to those issues relies on separating the merge driver from subsequent conflict resolution tools or 'merge tools'. The diff3x format proposed here would work well for data exchange between these two steps. This is important because it separates knowledge of the format and structure of a file from a GUI designed to allow the acceptance of changes or the resolution of conflict thus enabling a single GUI to handle many different file structures in a consistent way.

The intention of diff3x is to provide a richer format than diff3 for the exchange of differences and conflicts in text files. The advantages of diff3x over diff3 include the following:

- 1.** Recording user selections of options: this provides both the facility to save changes during a merge resolution session and providing an audit record of the choices made.
- 2.** Connected options: this provides the ability for the selection of one option to trigger the selection of another connected option, e.g. when a start tag is selected the corresponding end tag is also included.

3. Auto include of text: this provides for certain text to be automatically included if certain options are selected, e.g. so that appropriate separators could be included.

We have shown that improvements to the representation of change for structured data is possible and desirable. Changing the existing diff3 format proved to be awkward and limited, so this move directly to a markup representation using XML is a better approach. In working this through in more detail in this paper we have identified other advantages to make it easier for users to accept or reject changes in a way that is consistent with the underlying structure of the data.

References

- [1] *An Improved diff3 Format for Changes and Conflicts in Tree Structures.* [<https://markupuk.org/2019/Markup-UK-2019-proceedings.pdf#page=87>]
- [2] *GNU DiffUtil - Comparing and Merging Files.* [<https://www.gnu.org/software/diffutils>]
- [3] Khanna S., Kunal K., Pierce B.C. (2007) A Formal Investigation of Diff3. In: Arvind V., Prasad S. (eds) *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science. FSTTCS 2007. Lecture Notes in Computer Science, vol 4855. Springer, Berlin, Heidelberg* [https://link.springer.com/chapter/10.1007%2F978-3-540-77050-3_40]
- [4] *Schematron QuickFix* [<http://www.schematron-quickfix.com/>]
- [5] Robin La Fontaine and Nigel Whitaker *Merge and Graft: Two Twins That Need To Grow Apart* [<http://archive.xmlprague.cz/2019/files/xmlprague-2019-proceedings.pdf#page=175>]



On the Design of a Self-Referential Tutorial

Steven Pemberton, CWI, Amsterdam

XForms [XF1][XF2] is a declarative programming language, with a different programming paradigm compared to traditional programming languages. Since it requires a new way of thinking about programming, learning materials are paramount for people wanting to use the language and understand its benefits.

This paper discusses the method used to build a hands-on tutorial for XForms, using XForms, with the decisions taken, the techniques used, and the benefits that the approach gave.

1. Tutorials

A tutorial is a challenging educational setting, both for those learning and those instructing. Time is severely limited, and there is inevitably far more material on the subject than can possibly be taught in the time available. Unavoidably, compromises have to be made: do you make it deep and narrow, where you learn a subset of the material in great detail, or broad and shallow, where you get an overview of the whole, with less detail? Should it be a teaser to tempt attendees to later self-study, or a starter, so that attendees have at least a working knowledge of some of the material?

And then there is the method of presentation: should it be lecture style, permitting coverage of more of the material, or hands-on, allowing attendees more direct acquaintance with the material, while reducing the coverage?

1.1. XForms Tutorials

Most XForms tutorials [e.g. t1, t2] had to-date been of the lecture style, covering most of the language. However, a request for a hands-on tutorial motivated a new approach.

After attending several hands-on tutorials on other topics, the author concluded that in order to optimally use the time available for exercises, attendees shouldn't be required to start from scratch, since precious time is lost dealing with trivial issues like set up and syntax. Rather, the exercises should all require the attendee to make a change to an existing, working, example, using the newly-acquired knowledge. In that way, you get the advantage of the hands-on approach, while minimising trivial administrative details, with the added advantage of attendees being confronted with larger working examples right from the start.

As a consequence, the tutorial was designed as a rapid-fire sequence of exercises, each consisting of 5 minutes of presentation, followed by 5 minutes of coding. This has the added advantage of maximising the attendees' concentration, thanks to the recurrent switching of activities. The exercises themselves are mostly not stand-alone, but cumulative, each one building on an earlier one, so that at the end the attendee has a handful of small, but in themselves useful, applications. Although the tutorial was designed to be part of a live event, it also supports the use of self-study.

The resulting tutorial [t3] is interesting in that it is not only *about* XForms, but is also built *in* XForms, which in itself gave surprising possibilities.

2. Content

The first task in designing a tutorial is to identify the topics to be taught. This was a fairly easy job in this case, since it only involved running through the headings of the XForms specification, and selecting topics. Naturally enough, a structured XML document listing those topics, and their relative structure was created as part of this process:

```
<learn title="What you will learn">
  <topic name="Structure of an XForm">
    <topic done="" name="&lt;model&gt;"/>
    <topic done="" name="content"/>
  </topic>
  <topic name="Instance data">
    <topic done="" name="internal &lt;instance&gt;"/>
    <topic done="" name="external &lt;instance @resource&gt;"/>
    <topic done="" name="&lt;bind&gt;">
      <topic done="" name="@ref"/>
      <topic done="" name="@type"/>
      <topic done="" name="@calculate"/>
      <topic done="" name="@relevant"/>
      <topic done="" name="@required"/>
      <topic done="" name="@readonly"/>
      <topic done="" name="@constraint"/>
    </topic>
  </topic>
  ...

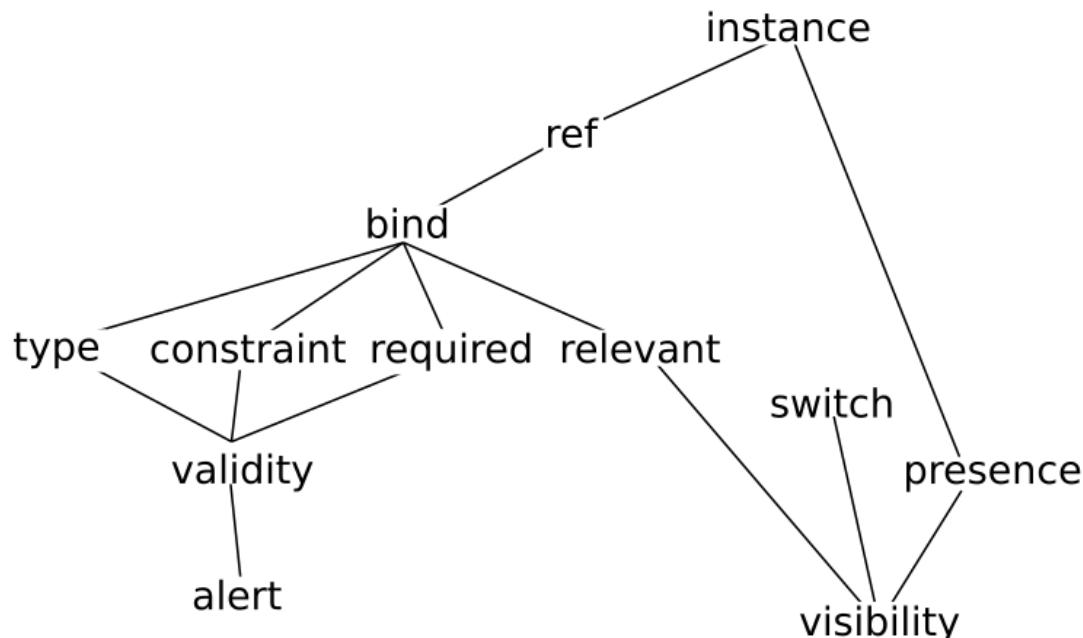
```

In fact this one document ended up being used in three ways: firstly and primarily as the list of topics that needed to be covered, that could then be checked off one by one as the tutorial was written, as a check for completeness. But additionally it ended up displayed in two different ways in the tutorial itself: once at the beginning as an overview of what would be taught in the course, and again at the end of the course, though displayed in a different way, as a summary of what had been taught, allowing the reader to check each topic off as a reminder of what had been learned (a correspondent had indicated that tutorials that included such a check-list on average were evaluated by attendees 10% higher). In fact this last section, allowing the attendee to check off topics learned, was the first written, so that as the tutorial was developed, the topics could be checked off.

The next step is to decide the *order* these topics need to be taught. This was done by creating a dependency graph: to understand this topic, you need to know about these topics, to understand *those* topics, you need to know about these other topics, and so on. This gives a partial ordering of topics, which can then be ordered linearly at will.

For example, to understand the `alert` element, you need to know about validity; to understand validity, you need to know about type, constraint, and required, to know about those, you need to know about bind, and ref, to know about those you need to know about instance.

Figure 1. A simplified partial dependency graph



It is worth pointing out that there is some leeway in even the partial ordering that the dependency graph generates, because rather than completely covering all the antecedents of a topic first, you can use progressive disclosure [pd]; for instance you could partially cover validity, talking only about types, and then extend it later with constraints and required. What the graph does reveal though is the order the partial disclosure has to happen.

This then leads to a fundamental decision: should the topics be taught top-down, or bottom-up? Bottom-up starts at the leaves of the dependency graph and works upwards building higher abstractions out of the lower-level ones; top-down works in the other direction, starting with the high-level abstractions, and working down towards the leaves. I am not aware of studies of the relative effectiveness of the two methods, nor whether one is more effective than the other with certain types of audience. I personally believe, based on experience of teaching, that top-down works better in general, especially with knowledgeable audiences, since it gives more opportunity to see the wood for the trees: each new thing you learn, you understand how it fits in the larger picture.

3. Structure

The topics were largely treated one per chapter, and in general written in the order they would be presented. It was attempted to make each chapter not more than one screen-full long, around 50-60 lines of text, followed by its exercise.

Each chapter was built as a single XHTML+XForms document; if it became too long, it could be split into two chapters; occasionally, if a chapter was very short, or if context demanded it, two concepts would be covered in a single chapter.

An XML document was kept as chapters were written documenting the title, filename, concepts covered, name of the exercise file, and name of the example answer file:

```

<tutorial>
  <entry>
    <title>Introduction</title>
  
```

```
<file>intro.xhtml</file>
<concept>implementations</concept>
<concept>W3C standard</concept>
<concept>future</concept>
</entry>
<entry>
  <title>What you will learn</title>
  <file>learn.xhtml</file>
  <concept>quick reference</concept>
</entry>
<entry>
  <title>The structure of an XForm</title>
  <file>structure.xhtml</file>
  <concept>structure</concept>
  <concept>model</concept>
  <concept>instance</concept>
  <concept>controls</concept>
  <concept>XML</concept>
  <concept>media type</concept>
</entry>
<entry>
  <title>Data Instances and the Input Control</title>
  <file>input.xhtml</file>
  <exercise>input-exercise.xhtml</exercise>
  <answer>input-answer.xhtml</answer>
  <concept>&lt;instance resource="" /></concept>
  <concept>&lt;bind ref="" /></concept>
  <concept>@type</concept>
  <concept>integer type</concept>
  <concept>date type</concept>
  <concept>string type</concept>
  <concept>boolean type</concept>
  <concept>&lt;input ref="" /> control</concept>
  <concept>&lt;label></concept>
  <concept>@incremental</concept>
</entry>
<entry>
  <title>The output control</title>
  <file>output.xhtml</file>
  <exercise>output-exercise.xhtml</exercise>
  <answer>output-answer.xhtml</answer>
  <concept>&lt;output ref="" /> control</concept>
  <concept>@value</concept>
  <concept>XPath / @ * </concept>
  <concept>count() function</concept>
</entry>
...
...
```

4. Navigation

This document of chapters was then used as the basis of the navigation part of the tutorial. This was modelled on the navigation used in an earlier XForms application, the XForms 2.0 test suite [ts].

A navigation instance was created containing

1. a value base that defines where the chapter files are stored,
2. a value for the filename of the chapter that is currently being read,
3. a value that constructs the required URL from the base and the filename:

```
<instance id="nav">
  <nav xmlns="">
    <base>chapters/</base>
    <file/>
    <url/>
  </nav>
</instance>
<bind ref="instance('nav')/url" calculate="concat(..base, ..file)"/>
```

To set the filename, a `select1` control is populated using the tutorial chapters instance: the labels are the title of the chapter, and the value is the file name of that chapter:

```
<select1 ref="instance('nav')/file">
  <itemset ref="instance('tut')/entry">
    <label ref="title"/>
    <value value="file"/>
  </itemset>
</select1>
```

Either side of this control, are triggers that decrement and increment this value (with judicious use of preceding-sibling and following-sibling):

```
<trigger
  ref="instance('tut')/entry[file=instance('nav')/file]/preceding-sibling::entry[1]">
<label></label>
<hint><output ref="title"/></hint>
<setvalue ref="instance('nav')/file"
  value="context()/file"
  ev:event="DOMActivate"/>
</trigger>
```

In this way, the user can select a chapter either from the drop down, or by navigating from chapter to chapter using the arrows.

Selecting a chapter

► XForms Hands-On Tutorial

← Multiple instances →

- Introduction
- M What you will learn
- Y The structure of an XForm
- Data Instances and the Input Control
- The output control
- External Instances
- The repeat control
- Multiple instances**
- Selections
- To Dynamic Selections
- Events and actions
- Triggers
- Y Submission
- Submit error
- F Calculate
- If Attribute value templates
- The xforms-value-changed event
- Insert
- F Delete
- readonly

ce="currencies.xml"/>

, you use the instance() function with the id:
 "><label>Search</label></input>"

instance, which is always the default instance.

et of them by filtering:
 >...

contains(a, b)

Note that thanks to how the `ref` on the trigger works in XForms, in this case referencing the previous chapter, if there is no previous chapter, the trigger will not be displayed.

The initial chapter, with no previous

► XForms Hands-On Tutorial

Introduction →

Introduction

What you will learn

XForms is a declarative language for applications, on the web and elsewhere.

Who uses it

To display the content of the chapter, an html iframe is used:

```
<html:iframe id="chapter" src="{instance('nav')/url}"></html:iframe>
```

Thanks to how XForms works, whenever the `file` value gets changed, the `url` value automatically gets updated, and so the new chapter is displayed.

The upshot of all this is that the tutorial instance doesn't *describe* the structure of the tutorial but actually *defines* it. If a chapter needs to be moved, you just move it in the tutorial instance; if a new chapter is written, add it to the tutorial document. The navigation will consequently automatically be updated, and the chapter can be displayed.

5. Chapters

Each chapter is thus a separate XForms document. This has amazing advantages when talking about XForms, because the text can actually contain the examples it is describing, rather than descriptions of the expected output.

For example, if a chapter contains the text:

By default, outputs directly after each other will abut. If a is 3, and b is 4, then

```
<output ref="a"/><output ref="b"/>
```

will output:

34

the underlying code is:

```
By default, outputs directly after each other will abut.
If a is <output ref="a"/>, and b is <output ref="b"/>, then
<group class="pre">
  &lt;output ref="a"/&gt;&lt;output ref="b"/&gt;
</group>
will output:
<group class="pre">
  <output ref="a"/><output ref="b"/>
</group>
```

In other words a really *is* 3, and b really *is* 4, and the example output is produced by the code itself.

This of course has immense advantage: you make fewer mistakes, you can put decisions off, or make changes easily without worrying about consistency between text and results.

6. Exercises

The exercises are based on 5 example applications: a short initial ‘toy’ one covering two exercises introducing the basic concepts such as instances, types, input and output, and then two large ones each covering 10 exercises, both broken in the middle by two small examples. The large examples helped with ordering the partial order of the material, since concepts could be introduced as they were needed for enlarging the functionality of the applications.

The first large example introduces external instances loaded over the internet, in this case the exchange values of a large number of currencies, and builds an application to display the exchange rate between two currencies of choice.

It is broken in the middle with a small application to show how times are manipulated, and to introduce the concept of events.

Submission is then introduced to show how to load new instances, and how to deal with submission errors, and then how to generate a URL for use in submission.

The second application is a to-do list, in order to show how to manipulate lists, how to use submission to save data, how to detect when data needs saving, and finally how to automatically save data when it has changed.

It is broken in the middle with a small log-in application, that shows how to input passwords, and how to deal with validity errors.

In some senses, finding suitable examples that were expandable in this way, gradually adding new concepts as they were needed was the hardest part of designing the tutorial. It is easy to create one tiny toy example for each concept being taught, but more work to

design ones that are meaningful and useful applications that have the right collection of necessary concepts.

7. Server

One of the requirements for following the tutorial is that the user have access to an HTTP server, and furthermore, one that supports the PUT method [put], since several of the exercises use it. For reasons that are not entirely clear, most HTTP servers available don't support PUT at all, or otherwise only with a lot of extra work. So the options for the tutorial apparently are: require the attendees to locate and install an existing server that accepts PUT; run a server and then require anybody who wants to use the tutorial to acquire a username and login for it; or supply a minimal server that the user can easily install and use. We opted for the third.

Luckily, writing a simple server without the normal industrial-strength requirements of large-scale servers is fairly easy, and since we had already written one for the XForms Test Suite [ts], XContents [xc], we used that, a few hundred lines of C code, which we then compiled for a number of standard platforms. Unfortunately, during testing, we discovered that on Windows, one virus scanner falsely identified the server as malware, and so for the people who ran against that problem we also wrote a JavaScript version which runs under Node.

Since the tutorial demands a server that uses PUT, the tutorial itself could use it for its own purposes. In particular whenever the user navigates to a new chapter, the current state is saved so that should the user stop and log out, on return the tutorial is restarted at the last-used location.

8. Experience

At the time of writing, the tutorial has been given once, not counting a couple of try-outs with single victims.

With only 2 hours allocated, and each exercise being budgeted for 5 minutes presentation and 5 minutes working, only 12 of the 25 sections were handled. Nevertheless, this wasn't seen as a problem, since the tutorial is also designed for self-study, and so the attendees could do the rest of the tutorial at their leisure.

There is a problem with doing a tutorial remotely, as was the case, that there is far less contact with the individual attendees, in order to solve problems, and less opportunity to assess if people have properly finished an exercise, especially since later exercises use the results of previous exercises. In fact this could be seen as a disadvantage of the cumulative approach to the exercises, in that it requires previous exercises to be completed before you can continue.

The tutorial was well-received; it revealed a few places where the exposition could be clarified, but otherwise went well.

9. Conclusion

Tutorial design, like any curriculum design, is hard! It is a surprising amount of work, with a lot of internal dependencies. However I have to admit that writing this one was a lot of fun, being about and in XForms allowed me to submerge myself fully in the subject, and allowed a lot of the internal dependencies to take care of themselves. Using XForms for the navigation introduced a large amount of flexibility in the construction of the tutorial.

Bibliography

- [t1] Steven Pemberton. *Declarative Applications with XForms*. XML Prague. 2020.
<https://homepages.cwi.nl/~steven/Talks/2020/02-13-xforms/> [<https://homepages.cwi.nl/~steven/Talks/2020/02-13-xforms/>] .
- [t2] Steven Pemberton. *Web Applications with XForms 2.0*. WWW 2013.
2013. <https://homepages.cwi.nl/~steven/Talks/2013/05-14-webapps-xforms2/> [<https://homepages.cwi.nl/~steven/Talks/2013/05-14-webapps-xforms2/>] .
- [t3] Steven Pemberton. *XForms Hands On*. Declarative Amsterdam. 2020.
<https://homepages.cwi.nl/~steven/Talks/2020/10-08-tutorial/> [<https://homepages.cwi.nl/~steven/Talks/2020/10-08-tutorial/>] .
- [xf1] John Boyer. *XForms 1.1*. 2009. W3C. <https://www.w3.org/TR/xforms11> .
- [xf2] Erik Bruchez et al. (eds). *XForms 2.0*. W3C. 2021. https://www.w3.org/community/xformsusers/wiki/XForms_2.0 .
- [put] T. Berners-Lee et al.. *Hypertext Transfer Protocol -- HTTP/1.0 (appendix D.1.1)*. . IETF. 1996. <https://tools.ietf.org/html/rfc1945#appendix-D.1.1> .
- [ts] Steven Pemberton. *The XForms 2.0 Test Suite*. MarkupUK. 2018. <https://markupuk.org/2018/webhelp/index.html#ar14.html> .
- [xc] Steven Pemberton. *XContents Minimal Web Server*. CWI. 2020. <https://homepages.cwi.nl/~steven/xcontents/xcontents.c> [<https://homepages.cwi.nl/~steven/xcontents/xcontents.c>] .
- [pd] Frank Spillers. *Progressive Disclosure*, in Soegaard and Dam (eds.), *The Glossary of Human Computer Interaction, Chapter 44*. The Interaction Design Foundation. undated. <https://www.interaction-design.org/literature/book/the-glossary-of-human-computer-interaction/progressive-disclosure> .



“FYI we’re not looking to go to print” Restyling the Markup UK Proceedings

Tony Graham, Antenna House

Markup UK is a markup conference, and its conference proceedings start life as DocBook XML markup. DocBook has a standard set of XSLT 1.0 stylesheets for transforming DocBook XML markup into other formats.

Markup UK 2018 was put together very rapidly, so it is a tribute to the usefulness of the DocBook XSLT stylesheets that the conference had proceedings at all. However, the PDF proceedings were the stock DocBook styles done with a Garamond font and with the addition of some front matter with sponsors’ logos and acknowledgements.

The Markup UK 2019 proceedings were produced using the same customisation. Happily for the conference, a second page of sponsors’ logos was needed, but that was about the only change. It was agreed shortly after the conference that the styles should be improved. The full instructions for what to do were “FYI we’re not looking to go to print, if that influences any of your decisions.”

It did, but it didn’t make the task any easier. This paper discusses the changes to the proceedings and how the DocBook stylesheets were customised to achieve them.

The ‘classic’ academic paper or conference proceedings paper (to the extent that there is one) has one or two columns of justified text per page. If there are two columns per page, the title and abstract, etc., typically span both columns.

Papers typically start on an odd-numbered, right-hand page: this could just follow from right-hand pages being the ‘front’ of a leaf, or it could be a by-product of single-article reprints of papers needing to start on a right-hand page.¹ Graphics are typically floated to the top (and sometimes also to the bottom) of the page. In a two-column paper, graphics can be either one column wide or span both columns. Large graphics might instead be grouped at the back of the paper, after the references. The combination of a two-column layout plus floating figures generally takes fewer pages than if all text, headings, and figures span the width of the page.

The changes fall into three areas: Markup UK look-and-feel; not going to print; and accessibility.

1. Markup UK look-and-feel

As stated previously, the 2018 proceedings used a Garamond font. It was initially intended to continue to use the Garamond in the restyling, but this Garamond has only regular and italic styles and does not have a bold weight. This was almost okay, because the heading hierarchy could have been indicated solely by changing font size, font style, and

¹Magazines and newspapers start articles as a two-page spread when it suits them, but in my limited experience, papers in journals and in conference proceedings do not start with a two-page spread.

spacing before and after titles. However, some of the papers used DocBook's `<emphasis role="strong">`, which is typically expected to render as bold text.

The Markup UK website has its own distinctive style, so the current PDF styles are based on that. Major headings use the same colour and 'League Gothic' font as the main title in the website, while minor heading and titles of tables and figures use the font with black text.

Figure 1. Website 'look-and-feel' applied to proceedings



The CSS for the website uses a 'font-family' property setting for body text that resolves as a Helvetica-like font on every platform:

```
font-family: "Helvetica Neue", Arial, sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "Segoe UI Symbol";
```

The proceedings instead use local copies of the open-source Liberation Sans font family [LIBERATION] so that PDFs produced on Windows and Linux are identical. The two members of the Markup UK committee who work on the proceedings use different operating systems that provide different fonts, so this is essential if they are to avoid seemingly random differences in their formatted proceedings.

Liberation Sans is a metrically compatible² open-source replacement for Arial [ARIAL], which has a proprietary license. Arial, which is packaged with Microsoft Windows, was created to be metrically identical to Helvetica so that a document designed for Helvetica could be displayed and printed without having to pay for a Helvetica license.

Liberation Sans has a complementary Liberation Mono font family that is used for program listings, etc. Liberation Mono is metrically compatible with Courier New, although its shapes are closer to Liberation Sans than to Courier New.

To save from having to configure the XSL formatter to use the fonts, the XSL-FO uses an AH Formatter extension for declaring local fonts inside `fo:declarations`:

```
<!-- https://github.com/liberationfonts/liberation-fonts/releases -->
<axf:font-face
  src="url('{$muk-xsl.dir}/liberation-fonts-ttf-2.00.5/LiberationSans-Regular.ttf')"
  font-family="Liberation Sans" />
<axf:font-face
  src="url('{$muk-xsl.dir}/liberation-fonts-ttf-2.00.5/LiberationSans-Bold.ttf')"
  font-family="Liberation Sans"
  font-weight="bold" />
```

The red highlight from the Markup UK logo is reused in list item markers and in callout numbers in a DocBook `<calloutlist>`.

²Corresponding characters have the same width.

Figure 2. programlisting callouts use Markup UK red

```
<animateMotion xmlns="http://www.w3.org/2000/svg"
  id="train.animation" xlink:href="#train" ❶
  begin="indefinite" fill="freeze" repeatCount="1" ❷
  calcMode="linear" keyTimes="0;1" keyPoints="0;1" ❸
  rotate="auto"❹
  dur="42.5" onend="eventEnded('train;section2.trail')❺>
<mpath xlink:href="#section2.path"❻/>
</animateMotion>
```

- ❶ The graphics group that will be subject to the animation
- ❷ Conditions for the start of the animation — in this case the animation waits until it is triggered explicitly. When the animation has finished freeze the graphics state, i.e leave the graphics translated to the end of the path and do not repeat.

List markers for <itemizedlist> are also diamond-shaped to reflect the overlapped '<>' diamond in the Markup UK logo.

Figure 3. List markers reflect the Markup UK logo

Example 1. Aggregated record as a hierarchy:

- ◊ travel id=t1
 - ◆ date: 2019-05-15
 - ◆ reason: Customer meeting in Oslo
 - ◆ employee name: Ola Nordmann
 - ◆ transactions
 - ◆ bus ticket - 100 NOK
 - ◆ accommodation - 1000 NOK

The grey rectangles with rounded corners from the website are reflected in the front cover title, paper titles, and as a border delimiting an abstract from the rest of its paper.

2. Not going to print

Because the proceedings were not intended to be printed, each paper starts as a two-page spread, with the first page as the left-hand page of the spread. This better suits modern wide displays that can easily show two pages side-by-side. The front matter also had several elements that started on right-hand pages. These can now start on either page, with no blank pages in the front matter.

There are also some things that were not done because of not going to print: for example, the cover image exactly fits its A4 page. If the proceedings were meant to be printed, the image would be larger than the page – it would ‘bleed’ past the edges of the paper – to avoid any misalignment in the printing and cutting leaving white edges on the paper.

2.1. Front matter

Figure 4. Original front-matter



The original front-matter comprised:

Figure 5. Restyled front matter



Changes to the front matter include:

The title incorporates the Markup UK logo (which has alternate text of “Markup UK”), and the year is coloured with the red of the logo. This was simple enough with XSLT 1.0, but it is the sort of thing that would be simpler with XSLT 2.0 or XSLT 3.0:

```

<xsl:variable name="text" select="normalize-space(.)" />

<xsl:choose>
  <xsl:when test="starts-with($text, 'Markup UK ')">
    <fo:external-graphic
      content-height="24mm" scaling="uniform"
      content-width="scale-to-fit"
      src="url({$muk-xsl.dir}/img/MarkupUK-2.svg)"
      axf:alttext="Markup UK" />
    <fo:block />
    <xsl:variable
      name="rest"
      select="substring-after($text, 'Markup UK ')" />
    <xsl:choose>
      <xsl:when
        test="string-length($rest) >= 4 and
              translate(substring($rest, 1, 4),
                        '1234567890',
                        '') = ''">
        <fo:inline
          color="{$muk.red}" text-depth="0">
          <xsl:value-of
            select="substring($rest, 1, 4)" />
        </fo:inline>
        <xsl:value-of
          select="substring($rest, 5)" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$rest" />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:when>
  <xsl:otherwise>
    <xsl:apply-templates />
  </xsl:otherwise>
</xsl:choose>
```

- ◊ Sponsor logos on two facing pages. The size and sequence of the logos were kept the same because it was not known what had been arranged with the sponsors about the appearance of their logos.
- ◊ Page of credits and thank yous. This, also, is largely unchanged from the original and for the same reason.
- ◊ Revised table of contents with formatting that reflects the formatting of the titles and author names in the articles. The titles are larger than in the original to make it easier to click on them to go to an article. The page numbers were kept small and are put close to the titles rather than being separated by leaders because the page numbers are less significant when the proceedings are read in a PDF reader.

2.2. Papers

Figure 6. Original paper

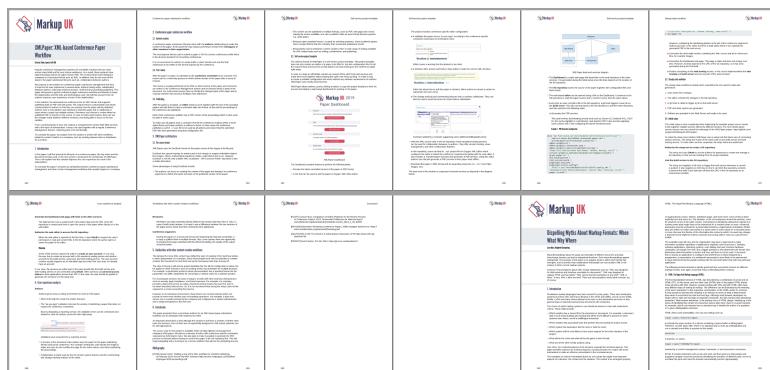


An original paper comprised:

- ◇ Title, authors' names and affiliations, and abstract (if present) on a right-hand page
- ◇ Text of the article, starting on the following left-hand page

The abstract page and the text pages are formatted as a single column with a wide left-margin. Section titles are formatted with zero left-margin so that they hang to the left of the body text.

Figure 7. Restyled paper



Changes to the formatting of pages include:

- ◇ Papers start on a left-hand page so that, on a wide screen, papers can be read as a sequence of two-page spreads.
- ◇ The title, authors' names and affiliations, and abstract (if present) are not on a separate page
- ◇ Left-hand and right-hand page are symmetrical, with a narrower margin next to the gutter and a wider margin on the outer edge of the page. The narrower margins are to make it easier for the reader's eyes to move from a left-hand page to a right-hand page.

3. Accessibility

Markup UK 2019 included the “Accessibility Matters” paper by Tony Graham. The slides, but not the paper, covered the facts that:

- ◊ Justified text causes problems for some people with certain cognitive disabilities
- ◊ Long lines can cause problems for people with some reading or vision disabilities
- ◊ People with some cognitive disabilities find it difficult to track text when the lines are close together.

The existing proceedings failed on every count except possibly the line height (but possibly only for some people). During and after the “Accessibility Matters” talk, there were multiple tweets from conference attendees about the difficulty of reading justified text. [LAPEYRE]

Figure 8. Tweet about justified text



Deborah A. Lapeyre
@dalapeyre

...
And another speaker comes out against JUSTIFIED TEXT, which is, as I was quoted as having said yesterday, "Evil", that is evil, evil, evil, and bad for dyslexics too #MarkupUK

10:56 AM · Jun 9, 2019 · Twitter Web Client

Given that, it was not credible to try for a ‘classic’ two columns of justified text with floating figures. Instead, the font size has increased – to reduce the number of characters per line – and the text is now left aligned. Hyphenation is disabled, so the only hyphens in the formatted text are from words, such as ‘over-enthusiastic’, that were written with hyphens. The proceedings now take about 10% more pages (currently 234 pages versus 212), but “we’re not planning on going to print”, so there’s no printing cost from the extra pages.

4. DocBook

As stated previously, the original proceedings used the DocBook XSLT 1.0 stylesheets. [XSLT10-STYLESHEETS] The restyling continued to use the XSLT 1.0 stylesheets because:

- ◊ There was an existing customisation for the sponsor logo pages and the credits page
- ◊ People were already familiar with using the customization (not least because at least one of them had written it)
- ◊ At the time, the DocBook XSLT 2.0 stylesheets had not gained much traction, and the XSLT 3.0 (xslTNG) stylesheets were not publicly available
- ◊ The initial approach had been to change very little, up until the Garamond font proved unsuitable and it was decided to be consistent with the Markup UK website

The style changes are made using the mechanisms that are provided in the DocBook stylesheets.

4.1. Titles

The DocBook XSLT 1.0 stylesheets support a templating mechanism for customising the title page, table of contents, and section titles. The template file is XML, of course. A template file is transformed into an XSLT module that is included when the DocBook source is transformed into XSL-FO. The template file is a mix of:

- ◊ Elements and attributes specific to the template mechanism

- ◊ Elements corresponding to the DocBook elements to be handled at that point
- ◊ Attribute values corresponding to XSL Formatting Objects to generate
- ◊ Literal XSL property attributes to generate in the output XSL-FO.

The templating mechanism does save a lot of work maintaining XSLT templates, but there is a learning curve to understanding the roles of the different elements and how they work together.

This is the beginning of the template for the front matter of an article:

```
<t:titlepage t:element="article" t:wrapper="fo:block-container"
    span="all" font-family="${title.fontset}"
    padding-bottom="1lh">
    <t:titlepage-content t:side="recto" start-indent="0pt" text-align="left">
        <t:wrapper t:wrapper="fo:block" background-color="${muk.background}"
            axf:border-radius="${muk.border-radius}" margin-top="20mm"
            margin-left="-150pt" padding-left="150pt" margin-right="0"
            padding="${muk.border-radius}" padding-bottom="0.25mm"
            axf:hanging-punctuation="start allow-end"
            color="${muk.blue}">
            <title t:named-template="component.title"
                param:node="ancestor-or-self::article[1]"
                keep-with-next.within-column="always" font-weight="normal"
                font-size="30pt"/>
            <subtitle font-size="20pt"/>
        </t:wrapper>
        <productname param:node="ancestor-or-self::article[1]"
            keep-with-next.within-column="always" font-size="30pt"
            font-weight="bold"/>
        <corpaauthor space-before="0.5em"
            font-size="${author.font-size}"/>
        <authorgroup space-before="0.5em"
            font-size="${author.font-size}"/>
        <author space-before="0.5em"
            font-size="${author.font-size}"/>
    </t:titlepage-content>
</t:titlepage>
```

Elements and attributes with the `t` namespace prefix, such as `t:titlepage`, are specific to the templating mechanism. The `template.xsl` stylesheet in the DocBook XSLT 1.0 distribution uses those to generate a stylesheet that can be included in the customisation. See Chapter 11, Title page customization, in *DocBook XSL: The Complete Guide* [DOCBOOK-XSL] for further information. Note that the `t:wrapper` element to wrap the definitions for other elements is an Antenna House extension that has only just been submitted as a pull request for the XSLT 1.0 stylesheets.

Elements in the default namespace have no namespace URI. These elements correspond to DocBook elements. Depending on a setting in the template file, the elements are processed either in the order shown in the template or in document order as they appear in the DocBook document.

The generated stylesheet includes an `xsl:template` for each of these elements. That template generates the element specified by the `t:wrapper` attribute of the `t:titlepage` element. Unprefixed attributes of the template element are copied as literal attributes on the wrapper element in the generated XSLT. The attribute values are not evaluated when the attributes are copied to the generated stylesheet, but they are evaluated when the generated stylesheet is used as part of a customisation, at which point any attribute value templates are evaluated. The generated stylesheet uses a sophisticated (or confusing,

depending on your familiarity with it) arrangement of attribute sets and mode names to make it possible to influence the processing of particular elements in specific or less specific contexts.

This is the template generated for author in the previous example:

```
<xsl:template match="author" mode="article.titlepage.recto.auto.mode">
  <fo:block-container xmlns:fo="http://www.w3.org/1999/XSL/Format"
    xsl:use-attribute-sets="article.titlepage.recto.style"
    space-before="0.5em" font-size="${author.font-size}">
    <xsl:apply-templates select=". " mode="article.titlepage.recto.mode"/>
  </fo:block-container>
</xsl:template>
```

When `t: named-template` is present, the generated template calls the named template instead of finding a matching template for the current element. Parameters to be passed to the named template are defined using attributes with the `param` prefix; for example, `param:node="ancestor-or-self::article[1]"`.

This is the template generated for title in the previous example:

```
<xsl:template match="title" mode="article.titlepage.recto.auto.mode">
  <fo:block-container xmlns:fo="http://www.w3.org/1999/XSL/Format"
    xsl:use-attribute-sets="article.titlepage.recto.style"
    keep-with-next.within-column="always" font-weight="normal"
    font-size="30pt">
    <xsl:call-template name="component.title">
      <xsl:with-param name="node" select="ancestor-or-self::article[1]"/>
    </xsl:call-template>
  </fo:block-container>
</xsl:template>
```

4.2. Other customisations

Other changes were made by overriding specific XSLT templates in the DocBook XSLT 1.0 stylesheets. This is fully expected for any non-trivial customisation of the stylesheets. The DocBook stylesheets are highly modular, and it was found to be simplest to put overrides in files corresponding to the original module containing the overridden template. For example, `muk-lists.xsl` overrides some of the templates in `list.xsl` in the DocBook XSLT 1.0 stylesheets.

4.2.1. Syntax highlighting

The customisation enables the option in the DocBook XSLT 1.0 Stylesheets to perform syntax highlighting on program listings using the XSLTHL syntax highlighting utility. [XSLTHL]

XSLTHL feels like a good fit for use with XSLT because it is an extension function for the XSLT processor, it does not require any other programming language, and its configuration files are XML. In reality: it is quite old; development on SourceForge had stalled and no new languages had been added for several years; it only works with Xalan, Saxon 6.5, and SaxonB (up to around Saxon 9.1) and only on Java.³

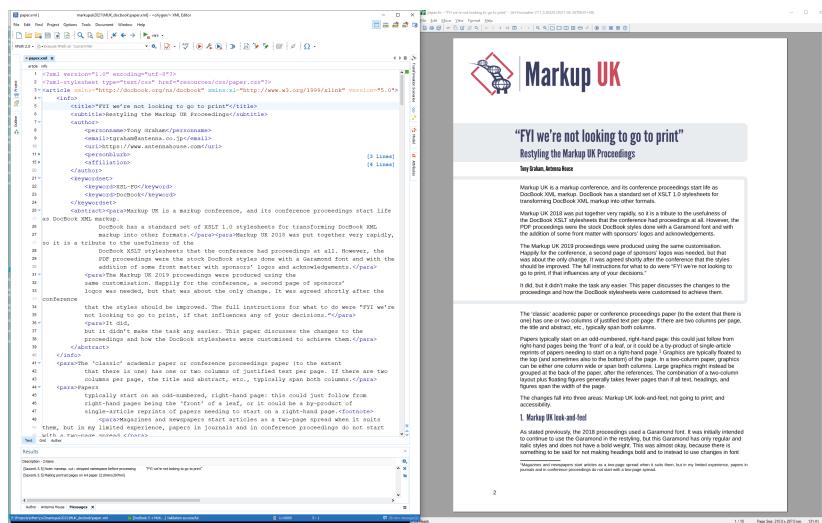
³Antenna House has contributed a PR that lets XSLTHL work with Saxon versions up to 9.9, but that is not in any official release.

Highlighters for XQuery, DTDs and batch files were added to the XSLTHL project on GitHub to handle more formats used in the papers.

5. Authoring

Issues with the 2019 authors' use of DocBook led to the development of an Oxygen framework that extends the DocBook 5 framework by adding some Schematron checks for some of the issues. The stylesheets for the proceedings are now also bundled with the framework so that authors can preview how their paper will look in the proceedings. The framework is available as an Oxygen add-on that the Oxygen editor will offer to update whenever a new add-on release is made.

Figure 9. Edit in Oxygen and preview in AH Formatter GUI



The support for authoring went through three stages:

- ◇ The extension framework and the Schematron file were available from the GitHub repository. [MUK-xsl] The Schematron was not used by the validation scenario defined in the framework.

Using the framework would require cloning the GitHub repository and configuring Oxygen to look for framework files in the cloned directory.

- ◇ The “markupuk-2019-paper” repository from Markup UK 2019 [XMLPAPER] was forked to create a new GitHub repository. The Markup UK stylesheets were added as a submodule, and transformation scenarios were added for formatting using the stylesheets. The new repository was configured as a “template”: forking a template repository on GitHub generates a new repository with all of the current files in the template but without their revision history.

Because the new repository does not share any history with its template, if the template repository is updated – for example, to include updates to the XSLT stylesheets – the changes cannot be merged into the new repository. If, instead, the new repository was a fork of the original repository, it would be possible to update the new repository to reflect changes in the original repository. However, if the user has modified their project file, there is potential for an update to cause conflicts that would have to be fixed by editing the framework XML file.

- ◇ The original framework repository was modified to include the stylesheets as a submodule and transformation scenarios to use the stylesheets were added. In addition,

the Oxygen add-on definition file that was already in the repository was modified to refer to a Zip file from the repository as the location for the add-on's files. For each new release, a new Zip file is generated and added to the files for the release. The add-on definition is updated to use the new version number and refer to the location of the new Zip file.

The framework now includes a template Oxygen project from which new Oxygen projects can be created. The template project includes a sample article that demonstrates some DocBook usage. It also defines some transformation scenarios, but these use the XSLT stylesheets from the framework rather than having the stylesheets bundled with each generated project. It is expected that the template project will not need to be updated very often. The most likely changes would be updates to the sample article, which would not affect an existing article.

Once Oxygen is configured to use the add-on definition, Oxygen will automatically detect new releases of the framework and offer to update it. This saves authors from needing to think about Git and Git submodules. If the template project is updated, the only way to get the changes is to create a new project and then copy files between the old and new projects. However, as stated previously, the template project is expected to not change very much.

6. Current status

Overall, the new styles are an improvement on the existing styles. However, because the restyling happened after the 2019 conference and long after the papers were written (and because authors weren't provided with a sample of how to properly tag footnotes and references, etc.), the 2019 proceedings are in a limbo state where most of the remaining improvements require changes to the source that no-one has the time, energy, and authority to change. The authoring framework can help future authors, but it was developed too late to help the other 2021 authors.

The process of developing the new styles resulted in multiple improvements to the DocBook XSLT 1.0 Stylesheets and to the XSLTHL syntax highlighting utility.

Developing the customisation has been sporadic, and it is still a work in progress. The code is available on GitHub [MUK-xsl] for anyone to copy and modify. Comments and pull requests are welcome.

Bibliography

[DOCBOOK-XSL] Bob Stayton: DocBook XSL: The Complete Guide. Fourth Edition, September 2007, OASIS. <http://www.sagehill.net/docbookxsl/index.html>

[ARIAL] Arial. Wikipedia. <https://en.wikipedia.org/wiki/Arial>

[LAPEYRE] Deborah A. Lapeyre: Tweet. 9 June 2019, Mulberry Technologies, Inc. <https://twitter.com/dalapeyre/status/1137659806288924672>

[LIBERATION] Liberation fonts. Wikipedia. https://en.wikipedia.org/wiki/Liberation_fonts

[MUK-docbook] Markup UK: 'MUK_docbook' add-on Oxygen framework, Markup UK <https://github.com/MarkupUK/paperFramework>

[MUK-xsl] Markup UK: XSL stylesheets for Markup UK proceedings, Markup UK <https://github.com/MarkupUK/MUK-xsl>

[XMLPAPER] Cristian Talau: XMLPaper: XML-based Conference Paper Workflow. 6 June 2019, Markup UK. <https://markupuk.org/2019/webhelp/index.html#xmlpaper.html>

[XSLT10-STYLESHEETS] DocBook: DocBook XSLT 1.0 Stylesheets, DocBook <https://github.com/docbook/xslt10-stylesheets>

[XSLTHL] Michal Molhanec, Jirka Kosek, Michiel Hendriks: XSLT syntax highlighting, <https://github.com/xmlark/xslthl>



CSS From XSLT

Liam Quin, Delightful Computing

CSS Within is a new way to integrate CSS generation with HTML generation to help reduce or eliminate these problems.

The method used is to embed rule and media elements, in a CSS Within namespace, inside XSLT templates. This puts the CSS styles exactly where the XSLT developer needs them: right next to the XSLT instructions producing the elements that they style.

Limited usage experience suggests that CSS Within makes HTML generation easier to maintain and more robust over time.

1. Introduction

A common problem when using XSLT to make HTML and CSS is keeping the XSLT, HTML and CSS synchronized. Changes to the XSLT that affect what HTML is generated necessitate corresponding changes to CSS; changes to the CSS may necessitate changes to the XSLT. Over time there will be some elements in the HTML for which there is no style information in the CSS, or for which the style information is out of date; there will be rules in the CSS that are no longer needed, or, worse, that are out of date and are only sometimes triggered.

Generating CSS from within XSLT, perhaps with one large `xsl:text` element, helps to reduce the divergence by making it easier to find CSS rules that affect a given element. But it is still easy to forget to update the CSS. A search for `div.beer` might not show up anything, but the CSS might have a selector `fridge>*` which matches `beer` (or any other element) when it's in the `fridge`. So it is not always trivial to locate the appropriate CSS rule to update.

With XSLT 3 also comes `expand-text="yes"` and curly braces being potentially special inside text node constructors, conflicting with CSS syntax.

CSS Within supports multiple stylesheets (for example for Web and PDF), and can run in pure XSLT or with extension instructions currently available for Saxon, in Java). Limited usage experience suggests that CSS Within makes HTML generation easier to maintain and more robust over time. This paper describes how to use CSS Within in your own projects and also discusses the current implementations and their limitations.

2. Rules and Media

CSS Within introduces `css:rule` and `css:media` elements into XSLT stylesheets. We will begin by looking at `css:rule` since it is the fundamental building block both of CSS Within and, indirectly, of CSS itself.

Consider an XSLT template as follows:

```
<xsl:template match="products-found">
    <!--* match search results, if any *-->
    <div class="productlist">
        <xsl:on-non-empty>
            <h3>Products</h3>
        </xsl:on-non-empty>
        <xsl:apply-templates/>
        <xsl:on-empty>
            <p class="no-products">
                No products matched your irritating query. Go away.
            </p>
        </xsl:on-empty>
    </div>
</xsl:template>
```

Styling the *Products* heading might require CSS like the following:

```
div.productlist {
    border: 1px solid grey;
    padding: 1rem;
}
div.productlist>h3 {
    font-family: "Bland Sans", sans;
    border-top: 1px dotted grey;
}
```

This CSS fragment consists of two rules, each consisting of a *selector* followed by a group of rules contained within { curly brackets }. The first rule has a selector that says the rule applies to any *div* element whose *class* attribute contains the token *productlist*. The second rule has a selector that applies to every *h3* element whose immediate parent is such a *div* element. Assuming no other more specific or subsequent rule overrides these, the rules assign values to various CSS properties such as *padding* and *font-family*. Beyond this, the details of CSS are not important to CSS Within. However, it *is* worth noting that CSS uses a text-based non-XML syntax with which people who work with Web development are very familiar.

It is clear from this example that the link between the stylesheets and the HTML being constructed is fragile: if the generated element structure is changed, or the class names are updated, the original CSS selectors will no longer match. One way to mitigate this is to put the CSS rules right next to the place where the elements they govern are generated. To put the CSS *within* the template, we might combine them as follows:

```
<xsl:template match="products-found">
    <!--* match search results, if any *-->
    <div class="productlist">
        <css:rule match="div.productlist">
            border: 1px solid grey;
            padding: 1rem;
        </css:rule>
        <xsl:on-non-empty>
            <css:rule match="div.productlist>h3">
                font-family: "Bland Sans", sans;
                border-top: 1px dotted grey;
            </css:rule>
            <h3>Products</h3>
        </xsl:on-non-empty>
```

```

<xsl:apply-templates/>
<xsl:on-empty>
    <p class="no-products">
        No products matched your irritating query. Go away.
    </p>
</xsl:on-empty>
<div>
</xsl:template>

```

The CSS Within tool set includes XSLT that will read the stylesheet itself, extract all of the `css:rule` elements and write a CSS stylesheet file. In this case the CSS will be identical to that shown above.

What have we gained? First, we no longer have a conflict of syntax: there are no more curly braces and everything is in XML. Second, the CSS definitions are right next to the elements they style. It is easy to imagine changing the `h3` to an `h4` but forgetting to change the CSS file; even if we remember, we then have to open the CCSS file and *find* the right rule to change. But now the styles and the markup are in the same place we are likely to remember and, remembering, will of course find it easy to locate the style rule to update.

Sometimes you may generate the same structure from multiple places in your stylesheet, but of course you don't want to repeat the CSS rules in the generated stylesheet. In this case you can use an empty `css:rule` element and give it a `ref` attribute whose value matches the name attribute of another `css:rule` element. The name attribute on the `css:rule` element also serves as a reminder that the style might be used elsewhere, helping to avoid the situation where you accidentally delete a rule you thought you no longer needed.

If you are using CSS for both print and screen, or if you have stylesheets loaded only conditionally based on a media query (for example, containing extra rules for wide screens or overriding defaults set for circular displays such as on some wristwatches), you may well need to write out more than one CSS file with XSLT. In this case you can give `css:rule` elements a `stream` attribute, and the contents will only be included in the CSS ruleset you name. That way all the styles to do with a given output element are together and as easy as possible to update together, even if they are written out separately.

The way the CSS is written to files is described in a subsequent section in this paper.

If you have media queries in your stylesheet, you can use `css:media` elements to generate them; these elements contain `css:rule` elements:

```

<css:media when="min-width: 600px">
    <css:rule match="ul.letterindex">
        column-count: 2;
    </css:rule>
</css:media>
<css:media when="min-width: 800px">
    <css:rule match="ul.letterindex">
        column-count: 3;
    </css:rule>
</css:media>

```

3. The CSS Output

Alongside the generated HTML one needs of course to produce CSS. Current Web practice is to use a mixture of separately-served CSS resources and CSS styles embedded within

HTML itself. The primary tradeoff between using separate files and embedding is in bandwidth consumption when users visit multiple pages on the same site, or even the same page multiple times: a separate CSS file can then provide clear savings in bandwidth and in time to render the Web page. For the first time that a client loads a page, however, embedded styles for the first few elements that will be rendered on the page can reduce the time before the user can use the Web page. CSS styles embedded in HTML after the head, however, raise security questions outside the scope of this paper; we shall focus only on a separate CSS resource, created on the server as a standalone file.

In an XSLT stylesheet using CSS Within, you can call the `css:gather` template to produce the CSS; use this within an `xsl:result-document` element with `method="text"`. In the future there may instead be a `css:stream` element for this purpose.

3.1. The stream attribute

If you are writing to multiple CSS files, you can give each `css:rule` or `css:media` element a `stream` attribute; the value of this attribute can be `#all` or a space-separated list of identifiers (as for the HTML `class` attribute); the corresponding CSS rules are generated to replace a `css:stream` element having a `name` attribute with the same value; the stream name can also be passed as a parameter to the `css:gather` template which gathers the styles from where they occur in the stylesheet into one place. This is not currently implemented, but is planned.

4. Phases of Operation

When you run an XSLT processor on a stylesheet using CSS Within, the following operations must occur:

1. By default, `css:rule` elements are *literal element constructors*, meaning that your output will be littered with CSS. So this must be prevented;
2. Your stylesheet must be analyzed and all of the CSS elements gathered up into one or more CSS files.

4.1. Phase One: De-cluttering the Output

The first of these two phases can happen in one of three ways:

1. By pre-processing the stylesheet to make a copy without any `css:rule` or other CSS Within elements that would cause problems;
2. By post-processing the generated HTML to remove the `css:rule` elements that were spuriously generated;
3. By modifying the XSLT processor so that it ignores the content of `css:rule` elements.

The first option is easily done using `fn:transform()` and wrapper XSLT, although care must be taken to pass on any stylesheet parameters. However, it is not possible in general, because of stylesheet imports that might be conditional (depending on `use-when` expressions or XSLT 3 shadow attributes, for example), and because package import resolution happens dynamically.

The second option can however be done with `fn:transform()`, since the output documents generated by any `xsl:result-document` elements in the transform are not written out but are included in the map returned by `fn:transform()` and can be post-processed.

The author of this paper wrote both XSLT for option (2) and also a Java class that implements XSLT extension elements for Saxon which make `css:rule` and `css:media` simply return the empty sequence when the stylesheet is executed.

4.2. Phase Two: Writing Out the CSS

Again there are multiple approaches to finding all of the CSS elements and writing out the stylesheets. It is important that *all* of the styles are written, even if they are inside templates that were not matched, as they might be styles for dynamically generated content. However, because a stylesheet can be compiled, saved in compiled form, and executed at a later time, we cannot write a Java extension that saves all of the `css:rule` and `css:media` at compile-time in a singleton object that can then be returned by a `css:stream` instruction.

Currently, extraction of CSS styles is done with XSLT, and does not work with dynamic import or package selection. So far this has not been a problem in practice, and ongoing experiments with package conventions may in time provide a solution.

An approach using a template in each separate file or package that matches `css:gather` elements and then calls `xsl:next-match` seems very promising.

5. Pure XSLT Implementation

The pure XSLT implementation of CSS Within uses `fn:transform` to run the original transformation and post-processes the result to remove unwanted CSS Within elements. It then processes the stylesheet files themselves as XML documents to find all of the `css:rule` and `css:media` elements and construct stylesheets.

This approach does not work with separate compilation, as in that case the source code for the original transformation is no longer available. It also does not work with packages, since the package location mechanism is not visible at the XSLT level. However, it is still very useful.

6. Java Extension for Saxon

CSS Within currently includes a Java definition of XSLT extension elements for `css:rule` and `css:media` that simply return an empty sequence at compile time. When the stylesheet runs, the CSS Within elements are gone, so there is no need for a clean-up phase.

In development is an extension element for `css:gather` which at stylesheet compilation builds the CSS stylesheet so that when run, the `css:gather` element is replaced by an `xsl:text` element containing the text of the stylesheet. In order that the CSS fragments are collected from included and imported stylesheets and packages, `apply-templates` must be called in `css:gather` mode; the CSS Within stylesheet which can be imported contains a template that uses `xsl:next-match` to fetch the `css:gather` fragments from every other stylesheet module. Since all of this is arranged when the stylesheet is compiled, run-time support for the extension elements is not required. Each module must either include the CSS Within stylesheet, or must define a suitable `css:gather` template.

This `css:gather` implementation is then used by the public `css:stream` element to produce the text of the CSS stylesheet.

The complexities of configuring Saxon to run an extension, and of supporting separate compilation where the runtime engine is not necessarily in the same Java instance as

the execution, or even in Java at all, mean that at the time of writing this paper the implementation is not stable enough to redistribute, but the author expects that to change.

7. Discussion: Deployment

An unexpected benefit of having CSS inside XSLT templates is that you can copy snippets of XSLT from one project to another, perhaps to implement Web-page controls or other common functionality, and the style is carried with them too. You could also do this if you had the CSS *before* the individual XSLT templates instead of inside them, but in that case you would have the possibility of missing the styles for the first template, and either the template would be separated from any `xd:doc` elements preceding it, or the CSS would be before the `xd:doc` elements and hence separated too far from the body of the template to reap the benefits of CSS Within.

The CSS Within approach relies on the CSS cascade mechanism: it is because of this that is not necessary to specify full styles for each generated element. This also means that a generated element that can appear in multiple contexts may need multiple `css:rule` elements for the different contexts. As an alternative, the base styles can be placed in the template generating the ancestor element, and this relationship can be made specific with a `ref=` attribute on the individual `css:rule` elements to show the dependency. This makes no difference to the generated CSS, but helps the human maintainer of the stylesheet understand what has been done and keep it up to date. The XSLT implementation does currently check the `ref` attributes.

8. Limitations

Currently, the CSS Within implementations support neither separate compilation nor packages. This work is in progress at the time of writing this paper (April 2021).

Since there is no support yet for CSS at-rules such as `@font`, all such constructs must be placed in the header part of the CSS stylesheet. This precludes, for example, having differing font-face rules based on media queries (or complicates it). An alternative design was explored that involved giving `css:rule` elements attributes with the same names as CSS properties, such as `font-family='Times Roman', 'times', serif'`; this took CSS Within too far away from CSS syntax.

The CSS stylesheet is constructed from all of the `css:rule` and `css:media` elements in the order in which they occur in the stylesheet; there is no control over the ordering. It is not clear yet whether this is needed.

Finally, it is worth noting that CSS Within has only been used in single-person projects so far. Although the author took some trouble to keep the body of `css:rule` and `css:media` as plain text in CSS syntax, so that one can be in, so to speak, a CSS state of mind when reviewing the styles, the method has not been tested with different people working on CSS and on XSLT.

9. Future Work

Future work for CSS Within includes supporting more of CSS; today you can have a header, individual rules, and a footer; the header might for example define Web font rules. Adding more elements to support font and other at-rules directly might help maintenance of stylesheets.

You can reuse fragments of stylesheets with a name/ref mechanism, but this has not been explored extensively.

CSS Within could also do minification of CSS, and possibly automatic prefixing.

The most likely short-term enhancement is automatic generation of match patterns. Future work on enhancing the Java plugin to support automatically generating CSS files even in the presence of separate compilation is also planned.

10. Availability

CSS Within is (or will be) freely available on gitlab, and can be used professionally if desired. Comments and patches are of course welcome.

<https://gitlab.com/barefootliam/css-within>

11. Conclusions

The author has been using CSS Within for some time. The result is that going back to a stylesheet after some months to make a minor adjustment is now significantly easier, and even larger refactoring keeps the CSS and the HTML working together.

The work involved in refactoring an existing stylesheet into CSS Within should not be underestimated, although even in small projects bugs and inefficiencies are very likely to be found and eliminated as part of this process.

CSS Within is a useful way to manage CSS styling of XSLT-generated HTML or XML in small projects; whether it scales up to larger multi-person projects remains to be seen.



Markup **UK**

<https://markupuk.org/>