Client-side XSLT, Validation, and Data Security

Wendell Piez

Abstract

Client-side XSLT (Extensible Stylesheet Language Transformations) or CSX is often used in scenarios where data (in XML, Extensible Markup Language) from a remote server is provided to a user who processes it in some way, for example rendering it locally for display. That is, the server provides the data, and the client does the work on that data to make it useful. However, that is not the only scenario in which CSX is useful. In an environment in which the user already has, or is in the process of creating, XML, CSX can be a convenient and powerful tool, enabling users to perform operations on their data securely on their own systems. The potential for this use of CSX is illustrated with uses of SaxonJS for several security-related applications.

Table of Contents

Context: CSX

As an acronym denoting “client-side XSLT”, “CSX” is offered in this paper in order to distinguish the focus from several closely related topics. It is a recognizable technology we have discussed – and demonstrated – for over two decades. Having an acronym for it distinguishes it (as an architecture and technology stack) from (on the one hand) broadly related topics such as XSLT (eXtensible Stylesheet Language) or declarative markup (see Declarative Markup bibliography) or (on the other hand) particular implementations such as (today) SaxonJS (Saxon JS). This is worth stressing because, while the projects described here are necessarily dependent on the stack on which they are built and deployed (in this case both SaxonJS and Github Pages), the questions I am seeking to pose are more general and more open and, hence, are easily confounded and confused with broader problems and considerations.

In an effort to keep this focus, this treatment:

- Is not a discussion of the pros and cons of CSX (or SaxonJS in the browser). This topic is already amply covered in the literature.[1]
- Does not offer instruction or directions on “how to”.[2]
- Does not make a (factual) claim or set of claims; while there are experiments here, they are not well controlled, and their conclusions are difficult to test.

Instead, there are observations, implications and, speculations describing and presenting an exploration of an idea within a real-world context.

Two Questions

Any technology demonstration will make more or less sense depending on its audience's understanding of the context and purposes of use. To the eye, CSX looks “just like any web page”. Only architects and systems builders will typically know or care how information processing is distributed over a network. Yet what happens behind the scenes – in particular, which data sets are serialized (“written”) and transmitted where – can be very consequential. This paper and the demonstrations shown here are premised, it happens, on an interesting “special case”, namely what happens when data (specifically XML documents or data sets) to be processed belong not to the publisher but to the user; that is, the user is the consumer of their own data, the internet being used only as a delivery platform for the application and never a transmission medium for the data as such. This is in contrast to more familiar use cases, in which (however complex may be the transformation or the interactivity of presentation), the data to be viewed is still provided by the publisher (or other provider of the web page), not the user. The case where the user already has the data (and, indeed, might be tasked with evaluating it) is interesting not only because of its potential generality and ubiquity but because for many kinds of data – only one such example being data regarding system security and system security description – the user having the data may be more the rule than the exception. Do CSX applications prove to be advantageous in this kind of scenario?

Across application domains, the requirements we are confronted with are sometimes for very peculiar kinds of data processing with specialized data sets (see Lubell 2014, Piez 2018). The problem that arises is how to build, deploy and maintain systems that can meet these requirements and that are sustainable? In this context, XML and XSLT with their declarative foundations still offer the potential of transcending – while exploiting (it should be noted how they remain as dependencies in the foundations) – today's HTML/CSS/Javascript/JSON web.

> ***Note***
>
> “HTML”: Hypertext Markup Language; “CSS”: Cascading Style Sheets; “JSON”: JavaScript Object Notation.

In particular, today we see the intriguing possibilities of applications that go beyond rendering or display. In particular, CSX can in theory provide a basis for a distributed validation architecture supporting data exchange formats (especially open and non-proprietary formats) with “validation” here being defined loosely and widely to encompass any set of formal criteria subject to testing.

In this context, this paper is most interested in two questions at the intersection of many more:

- What kinds of use cases and applications can we see for CSX processing documents or data that are not “published” but produced locally?
- What happens as the web becomes a platform for encapsulated processing logic over declarative foundations, including logic devoted specifically to testing data integrity?

“Distributed validation” as described here is another idea that is by no means new or especially innovative. As an example, the W3C Markup Validation Service (for HTML and kindred formats) comes to mind. It is not even new for CSX.[3] Questions about the utility to users and communities of CSX-based validation, as balanced against other concerns such as ease of development or maintenance, remain. Does the flexibility and distributability of CSX make it well suited to address complexity at scale? One reason it is especially interesting – in light of the first question (what happens when the data belong to the user?) – is that the feature set of CSX as applied to “validation” very broadly (to include any analysis of fitness for processing) also appears to be a good fit for the requirements of the emerging domain of systems security-related data exchange.

Play-along demonstrations

The demonstrations are available at https://pages.nist.gov/oscal-tools/demos/csx/ (or in one case at https://wendellpiez.github.io/XMLjellysandwich/). As described above, all these demonstrations have in common one feature: they are designed to provide a useful application for users who bring their data with them. Unlike a publishing application (in which the content is provided along with formatting or rendering), these are applications where the assumption is that the user is already in possession of XML data that they wish to process.

> ***Note***
>
> Efforts will be made to keep the landing pages for these demonstrations stable and functional. The demos' longevity or lack thereof may itself prove to be something of a test.

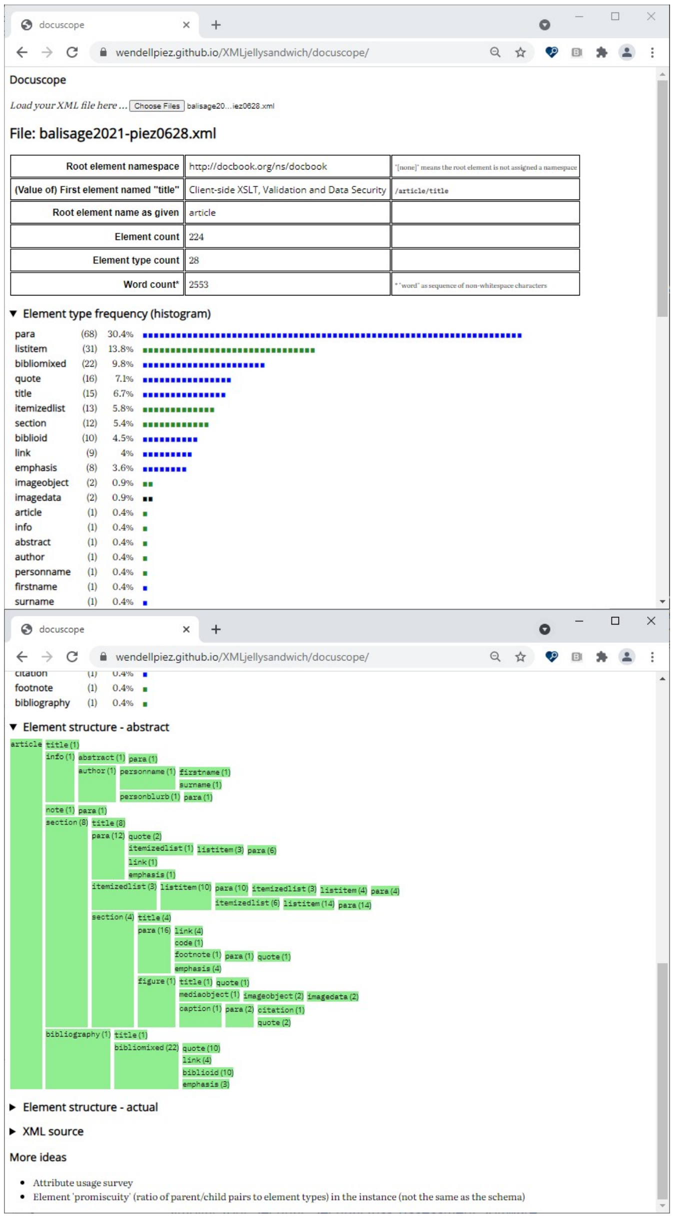
For those who do not have suitable XML readily available, a compressed (`zip`) file of examples is also provided (linked from that page) to use.

Docuscope

The demonstration may be loaded in a web browser using link https://wendellpiez.github.io/XMLjellysandwich/docuscope/.

This is a generic application designed to work on any complete and self-contained XML input.[4] It provides an analytic summary or synoptic view of the document, including showing its frequency of element (type) usage and mapping its abstract structure.

Figure 1: Two views of the docuscope (XML document inspection)



Docuscope

Load your XML file here ... balisage20...iez0628.xml

File: balisage2021-piez0628.xml

Root element namespace	http://docbook.org/ns/docbook	"[none]" means the root element is not assigned a namespace
(Value of) First element named "title"	Client-side XSLT, Validation and Data Security	/article/title
Root element name as given	article	
Element count	224	
Element type count	28	
Word count*	2553	* "word" as sequence of non-whitespace characters

▼ Element type frequency (histogram)

para	(68)	30.4%
listitem	(31)	13.8%
bibliomixed	(22)	9.8%
quote	(16)	7.1%
title	(15)	6.7%
itemizedlist	(13)	5.8%
section	(12)	5.4%
biblioid	(10)	4.5%
link	(9)	4%
emphasis	(8)	3.6%
imageobject	(2)	0.9%
imagedata	(2)	0.9%
article	(1)	0.4%
info	(1)	0.4%
abstract	(1)	0.4%
author	(1)	0.4%
personname	(1)	0.4%
firstname	(1)	0.4%
surname	(1)	0.4%
citation	(1)	0.4%
footnote	(1)	0.4%
bibliography	(1)	0.4%

▼ Element structure - abstract

```
article  title (1)
         info (1)  abstract (1)  para (1)
                   author (1)  personname (1)  firstname (1)
                                               surname (1)
                               personblurb (1)  para (1)
         note (1)  para (1)
         section (8)  title (8)
                      para (12)  quote (2)
                                 itemizedlist (1)  listitem (3)  para (6)
                                 link (1)
                                 emphasis (1)
                      itemizedlist (3)  listitem (10)  para (10)  itemizedlist (3)  listitem (4)  para (4)
                                                       itemizedlist (6)  listitem (14)  para (14)
                      section (4)  title (4)
                                   para (16)  link (4)
                                              code (1)
                                              footnote (1)  para (1)  quote (1)
                                              emphasis (4)
                                   figure (1)  title (1)  quote (1)
                                               mediaobject (1)  imageobject (2)  imagedata (2)
                                               caption (1)  para (2)  citation (1)
                                                                      quote (2)
         bibliography (1)  title (1)
                           bibliomixed (22)  quote (10)
                                             link (4)
                                             biblioid (10)
                                             emphasis (3)
```

▶ Element structure - actual

▶ XML source

More ideas

- Attribute usage survey
- Element 'promiscuity' (ratio of parent/child pairs to element types) in the instance (not the same as the schema)

The document being displayed is a draft of this paper.

It is easy to envision situations in which such a generalized "snapshotting" capability can be useful; alternatively, the approach could be extended to support very specific kinds of query, validation or analysis for particular document types or usage scenarios (as the following examples also illustrate).

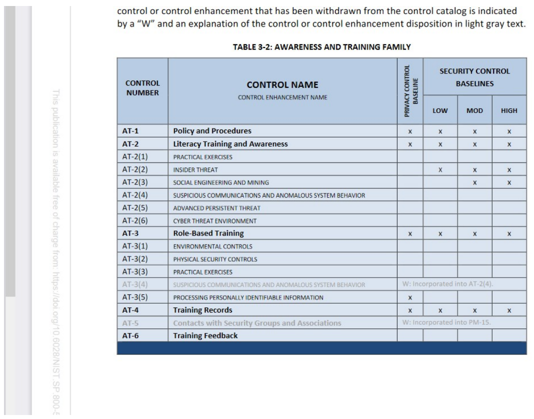
OSCAL baseline matrix

This demonstration may be loaded in a web browser using link https://pages.nist.gov/oscal-tools/demos/csx/baseline-matrix OSCAL Tools CSX Demonstrations. This and the following demonstrations described here assume XML documents conformant with OSCAL, the Open Security Controls Assessment Language.[5]

It loads an OSCAL `profile` (XML) document type and produces a formatted, tabular representation of its contents with a "look and feel" that emulates the official publication of analogous data sets.

For domain experts, this serves as an illustration of how, once requirements are clarified, automation can remove much of the most painstaking work of producing reasonably good production values in publication, thus (in this and potentially other ways) reducing significantly the effort (and cost) of developing and promulgating a baseline. This is, in part, because some of the more arduous operations are now automated and, in part, because editorial tasks and responsibilities are now decoupled from formatting, relaxing the need for close coordination between two “high touch” activities.

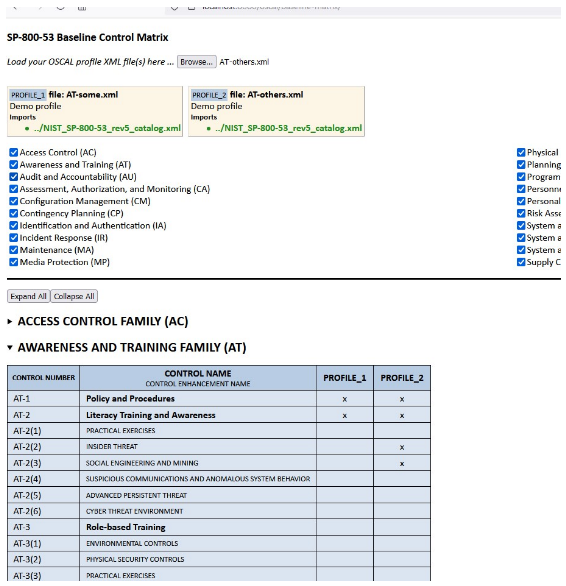
Figure 2: Emulating “look and feel” of an official publication



control or control enhancement that has been withdrawn from the control catalog is indicated by a “W” and an explanation of the control or control enhancement disposition in light gray text.

TABLE 3-2: AWARENESS AND TRAINING FAMILY

CONTROL NUMBER	CONTROL NAME CONTROL ENHANCEMENT NAME	PRIVACY CONTROL BASELINE	SECURITY CONTROL BASELINES		
			LOW	MOD	HIGH
AT-1	**Policy and Procedures**	x	x	x	x
AT-2	**Literacy Training and Awareness**	x	x	x	x
AT-2(1)	PRACTICAL EXERCISES				
AT-2(2)	INSIDER THREAT		x	x	x
AT-2(3)	SOCIAL ENGINEERING AND MINING			x	x
AT-2(4)	SUSPICIOUS COMMUNICATIONS AND ANOMALOUS SYSTEM BEHAVIOR				
AT-2(5)	ADVANCED PERSISTENT THREAT				
AT-2(6)	CYBER THREAT ENVIRONMENT				
AT-3	**Role-Based Training**	x	x	x	x
AT-3(1)	ENVIRONMENTAL CONTROLS				
AT-3(2)	PHYSICAL SECURITY CONTROLS				
AT-3(3)	PRACTICAL EXERCISES				
AT-3(4)	SUSPICIOUS COMMUNICATIONS AND ANOMALOUS SYSTEM BEHAVIOR	W: Incorporated into AT-2(4).			
AT-3(5)	PROCESSING PERSONALLY IDENTIFIABLE INFORMATION	x			
AT-4	**Training Records**	x	x	x	x
AT-5	Contacts with Security Groups and Associations	W: Incorporated into PM-15.			
AT-6	**Training Feedback**				



SP-800-53 Baseline Control Matrix

Load your OSCAL profile XML file(s) here ... Browse... AT-others.xml

PROFILE_1 **file: AT-some.xml**
Demo profile
Imports
- ../NIST_SP-800-53_rev5_catalog.xml

PROFILE_2 **file: AT-others.xml**
Demo profile
Imports
- ../NIST_SP-800-53_rev5_catalog.xml

▸ ACCESS CONTROL FAMILY (AC)

▾ AWARENESS AND TRAINING FAMILY (AT)

CONTROL NUMBER	CONTROL NAME CONTROL ENHANCEMENT NAME	PROFILE_1	PROFILE_2
AT-1	**Policy and Procedures**	x	x
AT-2	**Literacy Training and Awareness**	x	x
AT-2(1)	PRACTICAL EXERCISES		
AT-2(2)	INSIDER THREAT		x
AT-2(3)	SOCIAL ENGINEERING AND MINING		x
AT-2(4)	SUSPICIOUS COMMUNICATIONS AND ANOMALOUS SYSTEM BEHAVIOR		
AT-2(5)	ADVANCED PERSISTENT THREAT		
AT-2(6)	CYBER THREAT ENVIRONMENT		
AT-3	**Role-based Training**		
AT-3(1)	ENVIRONMENTAL CONTROLS		
AT-3(2)	PHYSICAL SECURITY CONTROLS		
AT-3(3)	PRACTICAL EXERCISES		

The second rendition here shows a display of two locally stored OSCAL `profile` instances (tagged as ***PROFILE_1** and* ***PROFILE_2***) loaded into and presented by the Baseline Matrix application. To generate the table, the user loads one or more valid OSCAL profile documents, which the browser then formats and displays, styling the data set as it would be shown in an official publication. The official publication in question (which offers this “model”) is NIST SP800-53B, as illustrated in the screen capture of the published PDF shown first.

As described in NIST Special Publication 800-53B, the security and privacy control *baselines* are predefined sets of controls (or their sub-items, control enhancements) specifically assembled to address the system and data security and protection needs of groups, organizations, or communities of interest. Each of the four control baselines defined in 53B is shown as a column in the table: HIGH, MODERATE, LOW and the supplementary PRIVACY baseline. (Of the core baselines, HIGH has the most controls and LOW the fewest, since HIGH is for systems that require higher security due to the nature of the data and systems, whereas LOW includes only the controls everyone should use. PRIVACY can be added to any baseline that needs it.) But as a starting point for the protection of individuals’ privacy, information, and information systems, each baseline is intended to be tailored further (i.e., customized), appropriately taking into account organizational missions and business functions, specific and credible threat information, the environment in which the organization operates, and individuals’ privacy interests. This tailoring takes the form first and foremost of adapting the control *selection* with reference to the catalog from which all the controls are derived; controls are removed from the baseline as inappropriate or inapplicable, or added. (Also see Lubell 2016 and Lubell 2020 for more work in this area. Joshua Lubell provided the idea for this demonstration.)

In the official publication, the tabular display lists the controls and enhancements from the catalog, noting for each line item whether an official baseline includes it. But when organizations have their own baselines (which remove or add controls as appropriate), the official tables are no longer applicable to them. OSCAL represents control baselines as OSCAL “profiles”; given such a profile (in XML), this application makes the table. Under CSX, the dynamic view can be produced for any profile and can be refreshed as needed.

OSCAL profile import examiner

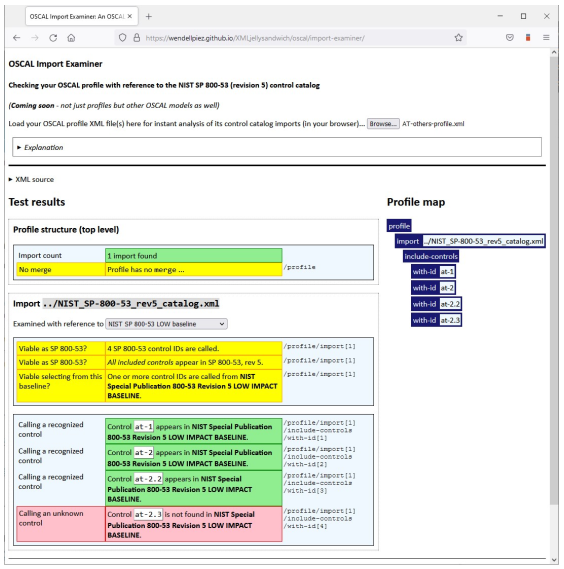
The demonstration may be loaded in a web browser using link https://pages.nist.gov/oscal-tools/demos/csx/import-examiner OSCAL Tools CSX Demonstrations.

This application provides an example of an application-specific validation, in the sense that it encapsulates a specific set of rules that together address some set of functional requirements (for the data in question), without addressing larger questions of formal validation to an abstract model.[6] The point here is to use the validation in question to test the fitness of the document for (some kind of) processing, independently of and complementary with a validation of the same document against any appropriate schema(s).

A case in point is an OSCAL profile, which must be valid to the formal model defined for its document type, a set of rules that is expressed (for example) as an XSD (XML Schema Definition). However, in order to function as expected, a profile must be more than valid to its schema. In the case of a profile, for effectiveness (usefulness) in context, it needs to reference (`<import>`) another document – the OSCAL catalog – and, moreover, to reference the catalog's own contents at a granular level.

This application tests the integrity of the links in an XML document to assess specifically the fitness of any OSCAL profile (loaded by the user) to be treated as a *baseline* or *overlay* (see above and also Lubell 2020) of a particular catalog over and above its validity to its formal model (as an "OSCAL profile"). For reference, the application includes four widely used (even *de facto* canonical or "industry-standard") catalogs – namely, the full control set defined by NIST Special Publication (SP) 800-53, along with the three NIST SP 800-53B HIGH, MODERATE, and LOW security control baselines. The user can choose any of these as a basis for comparison (checking) of the loaded profile.

Figure 3: Broken import



OSCAL Import Examiner: An OSCAL

https://wendellpiez.github.io/XMLjellysandwich/oscal/import-examiner/

OSCAL Import Examiner

Checking your OSCAL profile with reference to the NIST SP 800-53 (revision 5) control catalog

*(**Coming soon** - not just profiles but other OSCAL models as well)*

Load your OSCAL profile XML file(s) here for instant analysis of its control catalog imports (in your browser)... Browse... AT-others-profile.xml

► *Explanation*

► XML source

Test results

Profile structure (top level)

Import count	1 import found	/profile
No merge	Profile has no `merge` ...	

Import `../NIST_SP-800-53_rev5_catalog.xml`

Examined with reference to NIST SP 800-53 LOW baseline

Viable as SP 800-53?	4 SP 800-53 control IDs are called.	/profile/import[1]
Viable as SP 800-53?	*All included controls* appear in SP 800-53, rev 5.	/profile/import[1]
Viable selecting from this baseline?	One or more control IDs are called from **NIST Special Publication 800-53 Revision 5 LOW IMPACT BASELINE.**	/profile/import[1]

Calling a recognized control	Control `at-1` appears in **NIST Special Publication 800-53 Revision 5 LOW IMPACT BASELINE.**	/profile/import[1] /include-controls /with-id[1]
Calling a recognized control	Control `at-2` appears in **NIST Special Publication 800-53 Revision 5 LOW IMPACT BASELINE.**	/profile/import[1] /include-controls /with-id[2]
Calling a recognized control	Control `at-2.2` appears in **NIST Special Publication 800-53 Revision 5 LOW IMPACT BASELINE.**	/profile/import[1] /include-controls /with-id[3]
Calling an unknown control	Control `at-2.3` is not found in **NIST Special Publication 800-53 Revision 5 LOW IMPACT BASELINE.**	/profile/import[1] /include-controls /with-id[4]

Profile map

profile
import ../NIST_SP-800-53_rev5_catalog.xml
include-controls
with-id at-1
with-id at-2
with-id at-2.2
with-id at-2.3

Where an OSCAL profile imports a catalog and references its controls, controls that are not found can be detected by the Import Examiner and displayed as errors or issues. In this case, the loaded document would come up free (no errors) when evaluated against the HIGH baseline, which comprises a larger set of controls than LOW and including `AT-2(3)`, the control imported here that shows up pink (as missing from LOW).

If a profile comes up clean (no issues) when examined in reference to the indicated catalog to be imported, the application reports it. Where there are issues, the Examiner also reports those. Such issues might include references to controls that do not exist; redundant or repeated references; contradictory or superfluous customizations of control contents (tailoring); and the like.

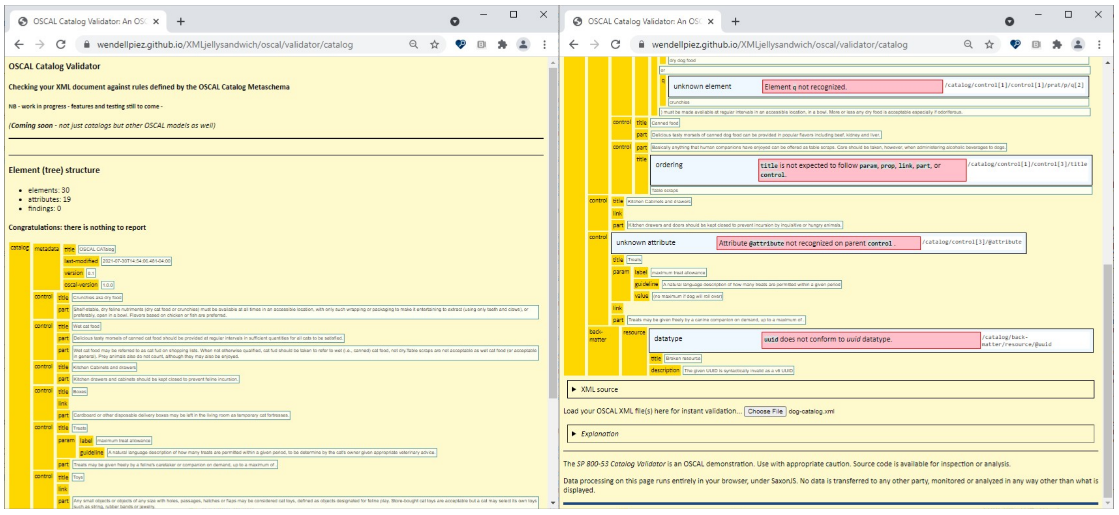
It is important to note that this is not an editor and does not offer the capability to create or change a profile. Rather, this application is meant to independently evaluate the correctness and conformance of tools and profiles produced by others, or as an independent check on the soundness of a profile still in draft.

OSCAL schema emulator

The demonstration may be loaded in a web browser using link https://pages.nist.gov/oscal-tools/demos/csx/validator in OSCAL Tools CSX Demonstrations.

In this case, the application offers the kind of top-down structural validation more usually provided by a conventional schema language. To provide this logic, the XSLT that is applied to the user's document has been generated directly from an appropriate OSCAL metaschema (a document type I described in Piez 2019), which is the same source from which the authoritative OSCAL XSD is generated. Accordingly, assuming this implementation correctly supports the semantics of the Metaschema language, it will report the same issues as XSD validation would.

Figure 4: Valid and invalid in the schema emulator


OSCAL Catalog Validator: An OSC

wendellpiez.github.io/XMLjellysandwich/oscal/validator/catalog

OSCAL Catalog Validator

Checking your XML document against rules defined by the OSCAL Catalog Metaschema

NB - work in progress - features and testing still to come -

*(**Coming soon** - not just catalogs but other OSCAL models as well)*

Element (tree) structure

- elements: 30
- attributes: 19
- findings: 0

Congratulations: there is nothing to report

catalog
metadata title OSCAL CATalog
last-modified 2021-07-30T14:54:06.481-04:00
version 0.1
oscal-version 1.0.0
control title Crunchies aka dry food
part Shelf-stable, dry feline nutriments (dry cat food or crunchies) must be available at all times in an accessible location, with only such wrapping or packaging to make it entertaining to extract (using only teeth and claws), or preferably, open in a bowl. Flavors based on chicken or fish are preferred.
control title Wet cat food
part Delicious tasty morsels of canned cat food should be provided at regular intervals in sufficient quantities for all cats to be satisfied.
part Wet cat food may be referred to as cat fud on shopping lists. When not otherwise qualified, cat fud should be taken to refer to wet (i.e., canned) cat food, not dry.Table scraps are not acceptable as wet cat food (or acceptable in general). Prey animals also do not count, although they may also be enjoyed.
control title Kitchen Cabinets and drawers
part Kitchen drawers and cabinets should be kept closed to prevent feline incursion.
control title Boxes
link
part Cardboard or other disposable delivery boxes may be left in the living room as temporary cat fortresses.
control title Treats
param label maximum treat allowance
guideline A natural language description of how many treats are permitted within a given period, to be determine by the cat's owner given appropriate veterinary advice.
part Treats may be given freely by a feline's caretaker or companion on demand, up to a maximum of .
control title Toys
link
part Any small objects or objects of any size with holes, passages, hatches or flaps may be considered cat toys, defined as objects designated for feline play. Store-bought cat toys are acceptable but a cat may select its own toys such as string, rubber bands or jewelry.

OSCAL Catalog Validator: An OSC

wendellpiez.github.io/XMLjellysandwich/oscal/validator/catalog

dry dog food
or
q unknown element | Element `q` not recognized. | `/catalog/control[1]/control[1]/prat/p/q[2]`
crunchies
) must be made available at regular intervals in an accessible location, in a bowl. More or less any dry food is acceptable especially if odoriferous.
control title Canned food
part Delicious tasty morsels of canned dog food can be provided in popular flavors including beef, kidney and liver.
control part Basically anything that human companions have enjoyed can be offered as table scraps. Care should be taken, however, when administering alcoholic beverages to dogs.
title ordering | `title` is not expected to follow `param`, `prop`, `link`, `part`, or `control`. | `/catalog/control[1]/control[3]/title`
Table scraps
control title Kitchen Cabinets and drawers
link
part Kitchen drawers and doors should be kept closed to prevent incursion by inquisitive or hungry animals.
control unknown attribute | Attribute `@attribute` not recognized on parent `control` . | `/catalog/control[3]/@attribute`
title Treats
param label maximum treat allowance
guideline A natural language description of how many treats are permitted within a given period
value (no maximum if dog will roll over)
link
part Treats may be given freely by a canine companion on demand, up to a maximum of .
back-matter resource datatype | `uuid` does not conform to *uuid* datatype. | `/catalog/back-matter/resource/@uuid`
title Broken resource
description The given UUID is syntactically invalid as a v6 UUID

▶ XML source

Load your OSCAL XML file(s) here for instant validation... Choose File dog-catalog.xml

▶ *Explanation*

The *SP 800-53 Catalog Validator* is an OSCAL demonstration. Use with appropriate caution. Source code is available for inspection or analysis.

Data processing on this page runs entirely in your browser, under SaxonJS. No data is transferred to any other party, monitored or analyzed in any way other than what is displayed.

The second document shows validation errors. (This is work in progress and user interfaces are rudimentary, but the errors here are correctly reported.)

This is of interest because (among other reasons) if XSLT can be deployed to support this kind of evaluation – in this case, to compile and deploy a higher order language describing constraints over data – a *schema language* – it can also presumably be deployed to support many other needs.

In other words, OSCAL Metaschema is not the only higher-order language that might be supported, in part or entirely, in an all-XSLT CSX deployment. In some ways this demonstration shows only that things can be done in XSLT that might be better done by other means and on other platforms. Yet the implications of that are profound. An application like this, which "only" emulates XSD, can also be a platform for another application that integrates this support with other functionality. It might reasonably be stipulated that such applications ought to be supported by schema-aware XSLT, without further layering or customization. And that is fair: we should be able to have either or both. In the CSX application we have "all XSLT all the time". Indeed, by demonstrating the same errors over the same inputs as an XSD processor (those inputs being valid or invalid), this application helps to corroborate (i.e., to validate at a higher level) a parallel XSD-based (or other) implementation of OSCAL, just as it is tested and corroborated in return, at least with regard to the supported features. The fact that the Metaschema language supports only a subset of XSD functionality (specifically with respect to content models) is helpful (see Piez 2019).

As the application here is provided by an XSLT programmatically generated from a Metaschema, it is relatively easy to envision other kinds of declarative languages that could be implemented via XSLT code generation and CSX-based deployment.

CSX for systems security applications

When we plan for everything to happen on the client system, we can deploy the application from a plain http (web) server with minimal or no special configuration. There is no need for a user's data ever to be exposed outside the local environment. The application having been delivered to the browser to execute, nothing is logged.

This raises some interesting questions about the special suitability or attractiveness of CSX for applications in which access, exposure, and confidentiality may be at issue. The domain of document exchange related to systems security is only one example where such requirements are common or prevalent.

Yet, even given this narrower scope, the questions here go beyond whether it is attractive to promote systems integrity and security by respecting – enforcing – system boundaries in application delivery and deployment (a question that would seem to answer itself). Once we can distribute not only data but capabilities, we can also envision decentralized networks not only of data exchange but of *validation* of artifacts of exchange. Are they what they say they are? Are they properly configured for use as expected? What features, usual or unusual, do they manifest? The problem here is not only how to exchange, and not even how to define exchange – to validate – but also how to ensure validability.[7] To be able to distribute application code to process data securely and independently has its attractions when we wish our data to interoperate without actually sharing it.

Specifically with respect to CSX, the same questions must be asked as we pose regarding any technology stack. What exactly is the security posture of SaxonJS and its dependencies (runtime and compile time)? Do we need a security assessment of SaxonJS? How about the XSLT that provides the actual logic to a CSX application and a critical point of exposure – how do we authenticate source and runtime (compiled) distributions? Surely the situation here is not worse than it is on the open web with Javascript libraries. But they are questions that should be asked precisely because the information custody model of CSX – “data are not exposed except to the user” – is so attractive.

Generalized capabilities

The good news here, especially for application domains that do not face the same challenges in either trust or access management, is that none of the XSLT methods or techniques used in any of those demonstrations (as opposed to the particular semantic exercised) are specific to the document type they address. It is quite easy to forecast how other domains and other types of documents present similar opportunities to their users, constituents, developers, and service providers; the hard work of formalizing these models provides a foundation for much more. The logic used to perform some of the OSCAL validations shown might be exactly the same as useful logic applying to TEI, DITA, or any of the commonly used XML documentary standards.[8]

Figure 5

```
<xsl:key match="person" name="persons-index" use="'#' || @xml:id"/>
...
  <xsl:if test="exists(key('persons-index',@ref,$prosopography))">...</xsl:if>
```

How a test for referential integrity might look over TEI (Text Encoding Initiative) - “does my content link to a person listed in my prosopography?”

```
<xsl:key match="party" name="parties-index" use="@uuid"/>
...
  <xsl:if test="exists(key('parties-index',@party-uuid,$authorizations))">...</xsl:if>
```

A similar test over OSCAL – “does my log contact info link to a party listed among authorized entities?”

These are essentially the same, once allowances are made for differences in naming and referencing syntax.

Another way of putting it is that while certain activities must be undertaken to understand these requirements in context, and to develop solutions to them, these activities are themselves intelligible and fairly well understood, and require and reward a set of common, transferable skills.

Things to do with your data

Considered generally, only a very brief and high-level survey is possible of the kinds of applications that might be deployed and maintained via CSX, with all the logic embedded in an XSLT and its contracts with a (more or less well defined) set of XML documents.

Distributed validation is one such application, as well as third-party validation – where the data are produced by A with tool A1, consumed by B with tool B1, but also validated by C. A concept of “micro-validation” is also conceivable, where full top-down validation of an XML instance, against the comprehensive rules of any nominal “type”, is not performed but only targeted validation of particular features related to particular rules and contracts.

Even more broadly, distributed data conversion capabilities are conceivable via CSX. Going beyond “transformation as a service” (a model in which the client sends the data up and receives a result back down), this would be "transformation as a product". As shown, example, perhaps a tool provides a preview more like the section “OSCAL baseline matrix” demonstration than like a formal validation – a preview being its own kind of validation. A crowd-sourced editing project might use such a tool to facilitate quality control. A conference might offer its authors a capability to preview papers as they would look in a published Proceedings. Authors then preview their own papers on their own systems without further effort on the publisher's part.

Expand this view even further and imagine micro-editors or customized, task-oriented interfaces – especially when these applications have “Save As” capabilities.

Additionally, it is not difficult to prognosticate possibilities from loading not one XML (or other) document but several at run time. One document could be a nominal “source”, while a second could be a set of rules expressed declaratively (such as a Schematron or something like it). Now the validation being performed is not simply hard-wired by the stylesheet, but instead dynamic, evaluating the user's document in light of the user's rules.

This opens the door to domain-specific languages of all kinds, both in XML applications and in applications that consume other formats. Given an XSLT-based parser implementation, a CSX application could read plain text – or any kind of specialized notation expressed in plain text – and produce markup.

Data description ecosystems

The wider implications would seem to be directly related to the potentials for well-structured, declarative formats as solutions to domain-specific problems. Distributed validation assumes that shared rule sets (whether or not described as “standards”) are in use, while it also stipulates that within the rules, actual data will be versatile and reusable over the long term in a wide range of *different* (while also consistent) applications. Otherwise, what is the point? Targeted validation strategies applied to specific use cases, application profiles, or even “best practices” descriptions, is one way of easing this tension.

It is not quite enough that rule sets can be exposed and shared as a “commons”. The declarative information models on which they depend must also be secure, which essentially also means standardized and supported by freely available public specifications. These are truisms within the markup community and among publishing technologists. Twenty years ago, we might have said “having standards and even commodity tools is not enough; we also need to work out how to share rules”. But they bear repeating at a moment when we are seeing *new* sets of rules emerge and become salient, even central, such as the peculiar sets of rules that relate to our data descriptions for security data (whether expressed in OSCAL or other formats). Together with and on the basis of the models and abstract specifications there is an entire ecosystem of tooling, both specialized and commodity tooling, that supports them and ultimately provides for the payback on our (considerable) investment, in the form of “meaningful outputs” or (put another way) “actionable knowledge”. A feature of client-side XSLT (CSX) is that it is encapsulated and manageable from an operational point of view – an important consideration in a “many-fits-all” operational scenario – while at the same time, it interoperates well – consuming the same data, executing the same paths – with other XML technologies on other platforms. If CSX is powerful, then CSX delivered by an XML database serving data over the web will be especially powerful. But even if you never connect the two, the simple fact that you can move your data across them makes them a powerful combination.

There are, it is true, concerns that also go with this higher level of question, such as validating the applications themselves. Some of those concerns were mentioned above (section “CSX for systems security applications”); more broadly, we can also stress that validation is finally done only in execution. This brings us back to fundamentals such as maintaining consistent inputs, replicating results, and publishing the results of testing openly. Any and all of the applications described in this paper might ideally (and maybe will) be supplemented with suites of unit tests that demonstrate the boundaries of their functionality in a way observable by any party. One pair of eyes is best checked with another.

Summary and Conclusion

In this paper I offered a new acronym, CSX, to designate an architecture, client-side XSLT, that I think rewards attention even while we have talked about it (in various implementations) for many years. Past demonstrations of CSX (including those offered by this author) have often proclaimed its potential, and in that respect, at least, we break no new ground: it *still* has great potential. The demonstrations offered and discussed here, however, seek to go at least one or two steps further to test more of these ideas in practice, by exploring real applications supporting operations that are actually meaningful to domain experts in the field.

The domain explored in these demonstrations specifically (namely, information system security as supported by the open, machine-readable formats now emerging to support data interchange among parties with interests at stake in this information) is both very specialized, and also not unusual in being specialized. One way it is special (but not unusual) in its requirements for information security; this is often information we would be happier not to send over the wire. CSX becomes interesting if it means that capabilities of *using*, *evaluating*, and *testing* the information come to us, while the information itself never leaves.

To the extent they are realized, the broader significance of these capabilities could go well beyond even the availability of tools for process improvement, because tools support standards even while standards support tools, and healthy standards will also enable new and better processes. In the case of CSX, we have a stack built on foundations in descriptive markup (XML) and declarative processing (XSLT), which (due to the web-based deployment model) conveniently requires little or no extra overhead to the end user, but which also (on the developer's side) comes ready-made to work with – integrate with and exploit – other related technologies. These shared foundations provide not only infrastructure but also a common base of knowledge, enabling solutions that are adaptable and responsive to new problems and new forms of old problems.

References

[Declarative Markup bibliography] “Declarative Markup: An Annotated Bibliography” See https://markupdeclaration.org/resources/bibliography.html.

[Delpratt and Kay 2013] Delpratt, O'Neil, and Michael Kay. “Interactive XSLT in the browser.” Presented at Balisage: The Markup Conference 2013, Montréal, Canada, August 6 - 9, 2013. In Proceedings of Balisage: The Markup Conference 2013. Balisage Series on Markup Technologies, vol. 10 (2013). https://doi.org /10.4242/BalisageVol10.Delpratt01

[Delpratt and Lockett 2018] Delpratt, O'Neil, and Debbie Lockett. “Implementing XForms using interactive XSLT 3.0.” Presented at XML Prague, https://www.saxonica.com/papers/xmlprague-2018ond.pdf

[Ford 2018] Ford, Katherine, and Will Thompson. “An Adventure with Client-Side XSLT to an Architecture for Building Bridges with Javascript." Presented at Balisage: The Markup Conference 2018, Washington, DC, July 31 - August 3, 2018. In Proceedings of Balisage: The Markup Conference 2018. Balisage Series on Markup Technologies, vol. 21 (2018). https://doi.org/10.4242/BalisageVol21.Thompson01

[Galtman 2020] Galtman, Amanda. "Saxon-JS Meets XSpec Unit Testing: Building High Quality Into Your Web App." Presented at Balisage: The Markup Conference 2020, Washington, DC, July 27 - 31, 2020. In Proceedings of Balisage: The Markup Conference 2020. Balisage Series on Markup Technologies, vol. 25 (2020). https://doi.org/10.4242/BalisageVol25.Galtman01.

[Lockett and Kay 2016] Lockett, Debbie, and Michael Kay. "Saxon-JS: XSLT 3.0 in the Browser." Presented at Balisage: The Markup Conference 2016, Washington, DC, August 2 - 5, 2016. In Proceedings of Balisage: The Markup Conference 2016. Balisage Series on Markup Technologies, vol. 17 (2016). https://doi.org/10.4242/BalisageVol17.Lockett01.

[Lubell 2014] Lubell, Joshua. "XForms User Interfaces for Small Arcane Nontrivial Datasets." Presented at Balisage: The Markup Conference 2014, Washington, DC, August 5 - 8, 2014. In Proceedings of Balisage: The Markup Conference 2014. Balisage Series on Markup Technologies, vol. 13 (2014). `10.4242/BalisageVol13.Lubell01`.

[Lubell 2016] Lubell, Joshua. "Integrating Top-down and Bottom-up Cybersecurity Guidance using XML." Presented at Balisage: The Markup Conference 2016, Washington, DC, August 2 - 5, 2016. In Proceedings of Balisage: The Markup Conference 2016. Balisage Series on Markup Technologies, vol. 17 (2016). `10.4242/BalisageVol17.Lubell01`

[Lubell 2017] Lubell, Joshua. "Using DITA to Create Security Configuration Checklists: A Case Study." Presented at Balisage: The Markup Conference 2017, Washington, DC, August 1 - 4, 2017. In Proceedings of Balisage: The Markup Conference 2017. Balisage Series on Markup Technologies, vol. 19 (2017). `10.4242/BalisageVol19.Lubell01`.

[Lubell 2019] Lubell, Joshua. "SCAP Composer: A DITA Open Toolkit Plug-in for Packaging Security Content." Presented at Balisage: The Markup Conference 2019, Washington, DC, July 30 - August 2, 2019. In Proceedings of Balisage: The Markup Conference 2019. Balisage Series on Markup Technologies, vol. 23 (2019). `10.4242/BalisageVol23.Lubell01`.

[Lubell 2020] Lubell, Joshua. "A Document-based view of the Risk Management Framework." In Proceedings of Balisage: The Markup Conference 2020. Balisage Series on Markup Technologies, vol. 25 (2020). https://doi.org/ `10.4242/BalisageVol25.Lubell01`.

[Lumley, Lockett and Kay 2017] Lumley, John, Debbie Lockett and Michael Kay. "Compiling XSLT3, in the browser, in itself." Presented at Balisage: The Markup Conference 2017, Washington, DC, August 1 - 4, 2017. In Proceedings of Balisage: The Markup Conference 2017. Balisage Series on Markup Technologies, vol. 19 (2017). https://doi.org/10.4242/BalisageVol19.Lumley01

[Maloney, Eaton and Beck 2015] Maloney, Chris, Alf Eaton and Jeff Beck. "A client-side JATS4R validator using Saxon-CE." Presented at Balisage: The Markup Conference 2015, Washington, DC, August 11 - 14, 2015. In Proceedings of Balisage: The Markup Conference 2015. Balisage Series on Markup Technologies, vol. 15 (2015). https://doi.org/10.4242/BalisageVol15.Beck01.

[OSCAL on the web] "OSCAL: the Open Security Controls Assessment Language." https://pages.nist.gov/OSCAL/ (accessed February and October 2021).

[Piez 2017] Piez, Wendell. "Interactive web applications: demonstrating SaxonJS." Presented at Balisage: The Markup Conference 2017, Washington, DC, August 1 - 4, 2017. In Proceedings of Balisage: The Markup Conference 2017. Balisage Series on Markup Technologies, vol. 19 (2017). https://doi.org/10.4242/BalisageVol19.Piez01.

[Piez 2018] Piez, Wendell. "Fractal information is." Presented at Balisage: The Markup Conference 2018, Washington, DC, July 31 - August 3, 2018. In Proceedings of Balisage: The Markup Conference 2018. Balisage Series on Markup Technologies, vol. 21 (2018). https://doi.org/10.4242/BalisageVol21.Piez01.

[Piez 2019] Piez, Wendell. "The Open Security Controls Assessment Language (OSCAL): schema and Metaschema." In *Proceedings of Balisage: The Markup Conference 2019*. Balisage Series on Markup Technologies, vol. 23 (2019). `10.4242/BalisageVol23.Piez01`.

[OSCAL Tools CSX Demonstrations] Piez, Wendell. "OSCAL Client-side XSLT (CSX) Demonstrations" (2021). https://pages.nist.gov/oscal-tools/demos/csx `10.18434/mds2-2490`.

[Saxon JS] Saxonica, Inc. Saxon JS landing page (with links to documentation). https://www.saxonica.com/saxon-js/index.xml.

[XSLT Fiddle] XSLT Fiddle. https://xsltfiddle.liberty-development.net/.

[1] See for example Saxon JS, Delpratt and Kay 2013, Maloney, Eaton and Beck 2015, Lockett and Kay 2016, Lumley, Lockett and Kay 2017, Delpratt and Lockett 2018, Piez 2017, Ford 2018, Galtman 2020.

[2] The site where these demonstrations were first developed, XML Jelly Sandwich (a personal project, by no means normative in any respect, and not endorsed by my employer), does include some instructional and helper materials, and is designed to be re-engineered and reverse engineered; but this paper does not discuss any of these technicalities.

[3] See Maloney, Eaton and Beck 2015.

[4] By “complete and self-contained” here we effectively mean it can be parsed by the library; unfortunately this means (in this case) DTD subsets and hence entity declarations will not be honored.

[5] See OSCAL on the web.

[6] In revision it occurs to the author that this might be termed “micro-validation” specifically to distinguish between this and the more generalized and totalized kind of validation we are more used to.

[7] Suites of unit tests could play a role in this context.

[8] DITA is the Darwin Information Typing Architecture; TEI is the Text Encoding Initiative.

Approximate word count: 5767. 5 figures.