# Relax NG schema for XSL-FO

Alexander Peshkov, RenderX Inc.

XML Europe 2004
18 - 21 April 2004
Amsterdam
Netherlands

## Table of Contents

### Abstract

In this paper a Relax NG schema for XSL-FO is presented. Advantages and drawbacks of the approach are also discussed.

## Introduction

There are no doubts about necessity and usefulness of document validation in the XML world in general, and in XSL-FOrmatting Objects in particular. However, the XSL-FO Recommendation includes no schema, and building one is not an easy task indeed. In this paper, we present a Relax NG schema for XSL-FO. We discuss various approaches to the formal description of XSL-FO document structure, with their relative strengths and limitations, and highlight sev eral intrinsic features of XSL-FO which have great impact on its formal schema. We conclude with pros and cons of the presented Relax NG schema.

## Approaches to XSL-FO formal description

The first and the most basic way of XML validation is by a DTD. RenderX created and maintained an unofficial DTD for XSL-FO for a long period: we used it for validation of the input stream in early versions of our XSL-FO formatter, XEP. DTD-based validation provides an easy way to check the structure of XSL-FO documents, especially since one doesn't need any special tools except a validating XML parser. DTDs are supported by a vast number of tools, and can be used both for input validation and schema-aided editing. But still, this approach is far from being perfect. General pros and cons of DTD validation has been widely discussed; we limit ourselves to point out several major limitations that are particularly serious for XSL-FO:

1. DTDs require an explicit reference to the DTD to be present in the document to be validated (as a system or public ID). This is very inconvenient in the case of XSL-FO documents generated dynamically - e.g. produced by an XSLT transformation;

2. DTD is not namespace-aware, while XSL-FO makes use of namespaces;

3. DTD cannot describe content models that depend on the tree context. XSL-FO uses context-dependent content models: allowed structure for a subtree may depend not only on ancestor elements but even on attribute values (outlines and absolutely positioned containers);

4. Element order in a mixed content model is not completely free in XSL-FO (e.g. "markers first" constraint); this cannot be constrained in a DTD;

5. Proper placement of properties and their values are hard to validate due to adoption of CSS inheritance model and use of expressions;

As you can see, some of these limitations are rather general while others are DTD-specific. There is a clear need for a tool that provides tighter control of XSL-FO document structure.

Use of rule-based language such as XSLT or Schematron can alleviate this problem and provide desired validation quality. This approach has been implemented as a dedicated XSLT stylesheet (folint.xsl), which is currently used in RenderX XEP to validate input. XSLT is a powerful language and allows to create a validator as elaborate as necessary. Besides the detailed validation, it provides a means to classify validity violations by severity, guarantees improved readability of messages and proper namespace support. Unfortunately, the solution also has considerable drawbacks:

1. Although XSLT processor is a common tool, the XSLT-based validation requires much more resources then the DTD-based one;

2. Since validation data are expressed in a programming language, they cannot be used as a schema for visual XSL-FO editors or document builders.

Yet another approach is validation based on an elaborate schema language. It can provide the necessary validation power while preserving high processing speed. Another important aspect is that a formal description can be used for schema-driven document creation by specialized software. There are two mature XML schema languages to consider: W3C XML Schema and ISO Relax NG. We preferred Relax NG, mostly because of simplicity and excellent supporting tools.

## XSL-FO modeling issues

First of all let's examine some XSL-FO intrinsic features that significantly complicate schema development.

When validating XML documents in general and XSL-FOs in particular we can separate three different aspects of a document's structure:

- Content models for elements;
- Constraints on attribute occurrence;
- Constraints on attribute values.

XSL-FO is a real challenge for schema writers, because all three aspects are interdependent: content models for some elements depend on values of attributes. Those elements are fo:block-container and fo:float.

Behavior of the first one is strikingly different depending on the value of its "absolute-position" property. Absolutely positioned block-containers (with `absolute-position="absolute"` or `absolute-position="fixed"`) resemble outline objects (floats): they are not allowed in the same contexts where outlines are not allowed, and cannot have outlines as descendants. Moreover, absolute and normal block-containers have quite different attribute sets. There is a clear need for separate element. In order to enforce XSL-FO requirements we were forced to create a special pseudo element in our schema – "`absolute-container`".

A similar problem arises with `fo:float`: depending on the value of the "float" property value, the element may assume two very different semantics (side-float and before-float), having quite different sets of constraints.

Note that both problems have the same origin – "float" and "absolute-position" properties were directly ported from CSS where they can be applied to the different elements. On the contrary, in XSL-FO those properties are applied to the dedicated elements only, and their semantics is overloaded (e.g. `<fo:float float="none">`).

Another architectural challenge imposed by XSL-FO is context-sensitivity of elements' content models. Although there is no notion of context for elements in XSL-FO spec, we can see at least three context flavors: inline-level, block-level and mixed. For example, inside `fo:block-container` or `fo:flow` there is a block-level context; inside `fo:block` – a mixed context; and inside `fo:leader`, `fo:title` or `fo:inline` child of `fo:footnote` (and their descendants except for `fo:inline-container`) there is a pure inline-level context. Note that the last context is inheritable.

Content model of some elements changes greatly in different contexts. The most evident example of such element is fo:wrapper and similar wrapper elements (`fo:multi-*` stuff). The inherited inline-level context is the most tricky since it actually necessitates maintaining separate content model for every element that can possible occur as a descend ant.

Note that XSL-FO do not define any common element content models – every element has a unique content model described using several "entities" ("`%block;`", "`%inline;`" etc.) together with a set of textual constraints. Elements of the same nature (and from the same group) still may have minor discrepancies in their content models that prevent re-use of the already defined patterns, making schema less transparent and more compicated. This effect is especially evident because of number of contexts mentioned above.

Inheritance and especially the use of expressions are the factors that make validation of properties placement unreliable. Even a property that is not inheritable nor applicable to the element can be purposefully specified on it in order to be retrieved later, e.g. using "`from-nearest-specified()`" function or special "`inherit`" keyword as the property's value.

Potential use of expressions also complicates strict validation of property values. Most of the properties have either a predefined set of possible values or some datatype restrictions, which can be thoroughly controlled by formal schema. However every property can also contain an expression. Since validation of expressions requires an expression parser and some non-local information, it is hard to implement it in a declarative schema language.

On the whole, during the development process we have discovered several issues and inconsistencies in XSL-FO specification, which were submitted to the XSL-FO working group.

## Relax NG schema design and limitations

Relax NG schema for XSL-FO has been created with two goals in mind: it should ensure fast and reliable validation; it should be convenient to use in visual XSL-FO editors or document builders. And, surely, we want a small, modular and easy-to-read schema. These criterions together with some XSL-FO features and Relax NG limitations are responsible for decisions made during development process. We discuss design of Relax NG schema in the rest of this section.

We have tried to formalize XSL-FO structure and construct schema from re-usable fragments. For this purpose we have explicitly separated various contexts described in the previous section and defined common content models and properties sets.

We followed the letter of the specification as close as possible. However, if we have to choose whether to implement some minor aspect of the spec at the cost of significant complication of the schema, we decided for less strict but simpler approach. A typical example is a requirement that element `fo:multi-toggle` should appear only as a descendant of `fo:multi-case` – this constraint was omitted due to its little practical meaning compared to required implementation efforts.

Let's recall property placement problem mentioned in previous section. From the formal point of view we have no choice but to allow almost all combinations of properties. However this will considerably decrease schema validation power and utility for visual XML editors. Therefore we resorted to the ad-hoc heuristics backed by common sense: an element may have all the properties which are applicable to it, as well as inheritable properties (or their subset), depending on elements' possible descendants. For example, it makes sense to allow all inheritable properties on fo:inline since its descendant can have a mixed content model, but block-level inheritable properties (such as margins) are meaningless when specified on `fo:page-number-citation`.

In order to validate property values we defined their patterns as a choice of several datatypes (including expressions). All datatypes and expressions (we treat them as a special datatype) were isolated in a separate module so that their definition can be easily customized.

Relax NG allows custom datatype libraries and have embedded support for W3C XML Schema Datatypes. Since most of the XSL-FO datatypes are fairly simple we were able to express reasonable constraints for them using basic means of WXS Datatypes – the only exception is XSL-FO expressions datatype.

Possible strategies of expressions treatment vary from the one that prohibits expressions and performs strict validation to "allow all" strategy. With limited level of strictness, expressions can be described using regular expressions available in W3C XML Schema Datatypes or provided by another datatyping method (such as Scheme language integrated in RNV Relax NG validator). Another option is to use dedicated XSL-FO datatype library – most of Relax NG tools provide interface for seamless integration of user libraries. However, development of such a library is a separate task and its value is to be determined.

Fortunately, Relax NG allows ambiguity for attribute values (in a choice of two datatypes both can match) so no matter whether expression datatype is restrictive or not, the schema will remain valid. This allowed us to leave expressions "underconstrained". Note also that Relax NG ambiguity is particularly useful for some properties (i.e. "`text-align`") that have choice of predefined tokens versus "`<string>`" value that is ambiguity in principle.

The only significant limitations of Relax NG expressiveness that we have noted during the development process is that Relax NG does not allow explicit exclusions at the element level and the only point where you can modify defin ition is inclusion of the separate Relax NG file. This problem was already discussed on Relax NG mailing lists but there is no sign that any of applicable solutions will be included in Relax NG in the nearest future.

Relax NG schema validation power is similar to the one of dedicated XSLT stylesheet ("folint.xsl") discussed before, but performance of Relax NG validation is higher even compared to the fastest XSLT engines. The only serious disad vantage of Relax NG validation compared to its XSLT counterpart is inability to classify validity violations by severity and lack of context-sensitive error messages. This is natural since Relax NG primary goal is to decide whether the supplied document is valid. However it's often helpful to have more detailed answer rather then "yes" or "no". Inability to segregate errors from warnings is especially important for XSL-FO – as we have shown above due to its somewhat intricate structure we may want to have several levels of validation strictness, i.e. report as warnings situations, which are valid from formal point of view but proven to be error prone.

This issue can be partially addressed with multiple validations. In such a case we have to create several customized versions of the same Relax NG grammar (according to desired strictness) and run validations in a pipeline. Hence vi olation of the "strict" schema should be treated as a warning while violation of the "loose" schema – as an error. Obvi ously, this approach introduce some redundancy but native efficiency of Relax NG validation and speed of available tools make even triple validation feasible.

Good tools is a weighty argument in favor of Relax NG. Relax NG is relatively young standard but there is a number of implementations out there already. First of all it is Jing – Java Relax NG validator written by James Clark. It is fast, reliable and has deliberate API. Another Java implementation is Sun Multi-Schema XML Validator created by Kohsuke KAWAGUCHI. Its Schematron add-on brings some interesting functionality lacked by Relax NG itself. Both these validators can be used as libraries and their API allows to use them as a SAX2 filters that is especially useful for multiple validation described in the paragraph above. RNV validator by David Tolpin is a pure C implementation of Relax NG which features extremely high-speed validation. There are also several XML editors providing support for Relax NG-based document creation such as Oxygen from SyncRO Soft or Topologi Collaborative Markup Editor. More tools can be found at Relax NG home page at Relax NG home .

## Conclusion

Formal description of XSL-FO document is not easy, but our experience has shown that Relax NG is up to the task. Being backed by OASIS and provided with a number of excellent tools, Relax NG has a bright future.

Relax NG schema beats DTD-based validation in every way. Compared to dedicated XSLT stylesheet it has less evolved error-reporting mechanism, but guarantees higher performance. Thus former approach is more appropriate for manual validation (where validation report will be consumed directly by user) while latter is more suitable for the batch validation or validation of automatically generated documents where speed is the main criterion.

In contrast to XSLT stylesheet, Relax NG schema can be used by advanced XML editors for the document building. DTD can serve the same purpose but with much less success due to its limited expressiveness. The only advantage of DTD in this context is a wider support in XML editing software. However the situation is changing with several Relax NG-aware XML editors already on the market. Additional argument in favor of Relax NG schema is that it can be used together with Namespace Routing Language to validate mixture of XSLT/XSL-FO during the development process.

We conclude that Relax NG schema is a fast and reliable tool for XSL-FO validation and it's a perfect base for schema-guided document creation.

Relax NG schema for XSL-FO presented in this paper is available as a public resource at xep.xattic.com (Resources Section) together with XSL-FO DTD and XSLT validation stylesheet (folint.xsl) mentioned above.

I would like to thank David Tolpin and Nikolai Grigoriev for their help and friendly guidance.

## Bibliography

[xep.xattic.com (Resources Section)] The Resources section of http://xep.xattic.com includes RelaxNG schema for XSL-FO, unofficial XSL-FO DTD and dedicated XSLT stylesheet for XSL-FO validation - folint.xsl.

[Relax NG home] The home page of Relax NG (http://www.relaxng.org) lists tools and documents related to this standard.