ONIX for Books 3.0: a ‘strict’ version of the XSD schema files

This document provides a description of some additions to the XSD schema for ONIX 3.0.4. The additions involve the creation of a parallel ‘strict’ version of the XSD schema using XSD 1.1 technology.

The ONIX 3.0 XSD schemas

The XSD schema is a definition of the structure of ONIX – it defines the tags that can be used, the way they are nested, optional or repeatable, and to some degree the data that each tag may contain. The XSD schema is used for validation – checking that an ONIX data file matches the specification.

But an XSD schema cannot check everything. For example, an XSD cannot do maths – it can’t check whether the check digit on an ISBN is correct. And it can’t check anything based on a data value elsewhere, so it can’t check a so-called co-occurrence constraint like ‘if it’s an e-book, it should not have a <Measure> composite’, or even ‘if the *dateformat* attribute is 00, the date must be in the form YYYYMMDD’. For the same reason, the XSD is unable to check on the use of codes from some of the codelists – 28 (in <Audience>), 98, 196, and others. And it cannot check on the rules around inclusion of <ROWSalesRightsType>. And the ONIX is lax about whether data elements can include extra leading or trailing spaces. There are many rules like this in the ONIX 3.0 *Specification* documents which cannot be enforced by the XSD schema.

At least, that’s true with XSD 1.0.

XSD 1.1 is a newer version of the XSD schema language – and it *can* do maths and check co-occurrence constraints, by setting rules or ‘assertions’ within the XSD file. These additional rules can in principle check that an ONIX data file matches the specification much more closely.

But XSD 1.1 is by no means ubiquitous. Some XML programming frameworks and tools don’t support it yet (including the popular libxml2 and xmllint). Because of this, the existing ONIX schema based on XSD 1.0 is not going to be *replaced* by a new schema based on XSD 1.1.

Finally, there are *several hundred* rules that could potentially be checked, enough that a new XSD 1.1 will never be complete – and some of the rules are *very* complex. So EDItEUR will take a gradual approach, releasing an initial experimental ‘strict’ XSD 1.1 that checks only a few key rules, and then adding extra rules later.

But it’s important that the strict XSD does not invent *new* rules and requirements. It checks only those rules that are explicit or clearly implicit in the *Specification* itself (including the codelists).

Using the strict XSD

Not all XML parsers support XSD 1.1. Parser frameworks like Saxon, Xerces and Raptor support 1.1, as do tools like oXygen or XML Spy that incorporate them. And with those tools, validating with XSD 1.1 works more or less exactly the same as with the original XSD 1.0. The only difference would be the filename of the XSD schema itself. However, the common libxml parser, tools like xmllint, and the standard XML parser within .NET do not support XSD 1.1 and cannot be used with the strict schema.

The beginning of an ONIX 3.0 data file – as it's normally transmitted – should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<ONIXMessage xmlns="http://ns.editeur.org/onix/3.0/reference" release="3.0">
```

(though the encoding might be different).

As part of a manual validation process, this usually gets modified to look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<ONIXMessage xmlns="http://ns.editeur.org/onix/3.0/reference" release="3.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.editeur.org/onix/3.0/reference
    ONIX_BookProduct_3.0_reference.xsd">
```

The additional *xmlns:xsi* and *xsi:schemaLocation* attributes associate the ONIX data file with the right XSD – the filename of the XSD schema is on the last line, in green. (NB this assumes the XSD schema is in the same folder as the ONIX data file itself, which isn't always the case, and the green text may include a longer path or even an http URL.)

To use the new strict schema, just add "_strict", so the green part ends with:

```
ONIX_BookProduct_3.0_reference_strict.xsd
```

and ensure the strict schema file is available in the same place as the ordinary XSD. Of course, if you use short tags, then you'll use:

```
ONIX_BookProduct_3.0_short_strict.xsd
```

That's it. If your XML validation software is compatible with XSD 1.1, validation will work as you would expect – and it will check various extra features of the ONIX data.

What rules are checked?

A set of more than 500 rules have been added so far. For example, in no particular order:

- the strict XSD checks that <RecordReference> is unique within the ONIX data file (*ie* that you don't have two product records for the same product in the same file)
- it checks the check digits on ISBN-13, GTIN-13, ISBN-10, UPC, ISNI, GLN, SAN, ORCID and ISMN-13 identifiers, and checks DOIs at least look plausible. It also checks that if you supply an ISBN-10, that the correct ISBN-13 is supplied alongside
- it checks dates are valid and match the *dateformat* or <DateFormat> specified. It also checks where date composites (eg <PublishingDate>) are repeated, that each has a unique 'role', and that for matching pairs of 'start' and 'end' dates, the end date is the later. Note that most dates must not be earlier than 1000 CE (anything earlier must be described as text using dateformat="12")
- if you use a proprietary identifier of any kind, it checks that you include <IDTypeName>
- it checks codes from 'second order codelists' like lists 28, 98 or 196 in elements including <ProductFormFeatureValue> or <AudienceCodeValue> that are not checked by the conventional XSD. Note that this requires the strict schema itself to be updated each time codes are added to any of the second order codelists
- it checks that subject codes from schemes including *Thema*, BIC, BISAC or CLIL look plausible (it does not check the subject code in detail, but ensures it conforms to the correct pattern for each scheme)
- it ensures that <Tax> detail is only provided for prices that are inclusive of tax. It also checks the tax calculation itself is consistent, and that <PriceAmount> is the sum of <TaxableAmount> and <TaxAmount>
- if you use contributor sequence numbers, it checks that there's one for every contributor and that they are consecutive integers starting at 1

- if you provide multilingual metadata by repeating a textual data element like <BiographicalNote>, it checks that every repeat has a unique *language* attribute (this also applies to other similarly-repeatable elements)
- if you repeat the <Extent> composite, it checks that each has a unique combination of <ExtentType> and <ExtentUnit>
- it checks that code 00 is not used in <SalesRightsType>, and that <ROWSalesRightsType> is provided when required
- it ensures that no country is listed more than once within elements like <CountriesIncluded> (and the same for regions, with the additional check that any <RegionsIncluded> that contains WORLD does not contain any other regions. Note that use of a couple of regions that are deprecated will make this check fail
- it checks that multiple <EditionType> codes are unique
- it checks that multiple <ProductContentType> and <PrimaryContent> codes are unique
- it enforces the rule that if and only if <BarcodeType> is 'not barcoded' then <PositionOnProduct> must be omitted, and it checks a similar rule linked to <PrintedOnProduct> within <Price>
- it checks for extraneous leading and trailing spaces in plain text data (though leading and trailing white space is still allowed in data fields that can incorporate XHTML

XSD validation is renowned for providing poor and sometimes misleading error messages, and so each of the new rules in the strict schema has an embedded 'friendly' error message, like this:

```
Assertion evaluation ('(: There must be an IDTypeName if (and only if) IDType
is proprietary :) (ProductIDType eq '01') eq exists(IDTypeName)') for element
'ProductIdentifier' on schema type '#AnonType_ProductIdentifier' did not
succeed.
```

Look for the messages between the `(:` and `:)` smileys. Note there are no friendly messages for the original XSD 1.0 rules that define the overall structure of ONIX.

Deprecation warnings

XSD rules are either pass or fail. However Schematron, a separate schema language, can provide *warnings*. With the strict schema, you can get the best of both. It has embedded Schematron rules that provide warnings when deprecated elements and codes are used.

By default, these Schematron rules are not enabled, and they make no difference to the usual XSD validation. To enable the deprecation warnings, an extra XML processing instruction needs to be added, like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model type="application/xml"
    schematypens="http://purl.oclc.org/dsdl/schematron"
    href="ONIX_BookProduct_3.0_reference_strict.xsd"?>
<ONIXMessage release="3.0" xmlns="http://ns.editeur.org/onix/3.0/reference"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ns.editeur.org/onix/3.0/reference
    ONIX_BookProduct_3.0_reference_strict.xsd">
```

The value of the *href* attribute in the new processing instruction (in green) matches the second part of the value of the *xsi:schemaLocation* attribute, and specifies the location of the XSD 1.1 schema file. (Some XML tools streamline the association of the schema with the ONIX file – adding the *xsi:schemaLocation* attribute and so on. The extra processing instruction to enable the embedded Schematron rules can usually be added in the same way.)

The Schematron warnings also include friendly error messages, such as "Warning: AudienceCode is deprecated".

Getting the strict schema, feedback to EDItEUR

The strict XSD schema files are downloadable from the EDItEUR website. If your validation framework is compatible with XSD 1.1, try it out – but note that although it has been in testing with a few ONIX users for more than two months at this point, the strict schema remains experimental – so don't try it on a production system.

And let EDItEUR know about the results. Does the XSD find errors in your ONIX data? Does it invalidate files that are in fact correct? What further rules would you most like to see added?

Graham Bell, EDItEUR
12th February 2018