EDItEUR

ONIX for Books

Product Information Message

Application Note: Twelve ONIX Problems – common issues faced by ONIX recipients

Recipients of ONIX data face many problems – the ONIX message format is relatively complex (because it aims to solve a relatively complex problem), and the number of potential data senders is large (particularly in markets where there is no strong central data aggregation service). But the difficulties can be exacerbated by needless variations in the way ONIX is used. These often need special post-processing of the data, manual intervention before received files can be processed, or manual correction of the data. ONIX is intended to facilitate automated exchange of large volumes of rich metadata, and any departure from the standard comes at a cost.

Many of the following issues were nominated by a major ONIX data recipient, based on the real-world ONIX it receives. But other recipients added to the list – and many attested to the fact that these issues are extremely common. The list originally appeared on the onix@groups.io mailing list (join the list by sending a blank message to onix+subscribe@groups.io).

ONIX senders – does your ONIX exhibit any of these issues? Remember, ONIX is just text, so as long as your ONIX file is fairly small, you can inspect it using a text editor or open it in a web browser.

12. Putting inappropriate information into title fields

– for example edition information or advertising text. Here are two examples, the first with an inappropriate subtitle and the second with unnecessary imprint information.

```
<TitleDetail>
       <TitleType>01</TitleType>
       <TitleElement>
              <TitleElementLevel>01</TitleElementLevel>
              <NoPrefix />
              <TitleWithoutPrefix>All for one</TitleWithoutPrefix>
              <Subtitle>The most explosive thriller you’ll read this year</Subtitle>
       </TitleElement>
</TitleDetail>
```

```
<TitleText>Anne of Green Gables (Munroe Classics)</TitleText>
```

In both cases, the extra information is useful, but should be in a dedicated field of its own – either as a Promotional headline (one of the many options in <TextContent>), or in <ImprintName> respectively. Many data recipients are forced to spend time taking this data out – time that could be spent on other improvements to the data.

11. Including invalid characters or using the wrong character encoding

– for example claiming the data uses UTF-8 when in fact it uses the Windows character encoding, or vice versa. Here’s an example of what can happen:

```
<PersonName>Marie-Adélaïde Barthélemy-Hadot</PersonName>
```

If the data sender gets the encoding declaration wrong, this name could be received as:

```
<PersonName>Marie-AdÃ©laÃ¯de BarthÃ©lemy-Hadot</PersonName>
```

This example shows ONIX that is really encoded as UTF-8 being received and interpreted *as if it were* Windows-1252. If the mistake were the other way around – it really is encoded as Windows-1252, but interpreted as if it were UTF-8, the entire ONIX file could be invalid.

In the example, the ONIX may well look correct to the sender, so why might a recipient have this problem? The first line of the ONIX (in fact of any XML file) declares the character set and encoding that the remainder of the file uses (or should be using). As a sender, you need to ensure that, in that first line, the encoding declaration actually matches the encoding of the file. It's not always UTF-8:

```
<?xml version="1.0" encoding="Windows-1252"?>
```

Your IT system will normally get this right, but if you cut and paste text from elsewhere into your system, it can go wrong. It's possible to check this by viewing a small ONIX file in your web browser.

10. Use of terms like N/A rather than omitting fields

Here's three different types of example:

```
<Contributor>
       <SequenceNumber>1</SequenceNumber>
       <ContributorRole>A01</ContributorRole>
       <PersonName>Various Authors</PersonName>
       <PersonNameInverted>Authors, Various</PersonNameInverted>
</Contributor>

<IllustrationsNote>N/A</IllustrationsNote>

<CityOfPublication><![CDATA[]]></CityOfPublication>
```

In the first case, you should use the <UnnamedPersons> tag, with code 04 to indicate 'various authors'. In the second and third cases, there's no information available so the entire field should be omitted. The last example is invalid.

9. Use of CDATA sections (*ie* using <![CDATA[…]]> when it isn't necessary)

This is all too common:

```
<PersonName><![CDATA[Catherine Holderness]]></PersonName>
```

The *only* time when it's necessary to use CDATA in ONIX is when the data itself contains HTML markup. In turn, that means CDATA should only ever appear in fields that can accept HTML.

These are the fields listed in the Specification as accepting HTML:

```
<AncillaryContentDescription>         <PrizeJury>
<AudienceDescription>                 <PrizeStatement>
<BiographicalNote>                    <PromotionCampaign>
<BookClubAdoption>                    <PromotionContact> (deprecated)
<CitationNote>                        <PublishingStatusNote>
<CopiesSold>                          <ReissueDescription> (deprecated)
<ConferenceTheme> (deprecated)        <ReligiousTextFeatureDescription>
<ContributorDescription>              <ReprintDetail>
```

```
<ContributorStatement>                 <SalesRestrictionNote>
<EditionStatement>                     <Text>
<EventDescription>                     <TextSourceDescription>
<FeatureNote>                          <TitleStatement>
<IllustrationsNote>                    <VenueNote>
<InitialPrintRun>                      <WebsiteDescription>
<MarketPublishingStatusNote>
```

CDATA in any other field is unnecessary, raises the potential for invalid data, and should be avoided.

Here's a more subtle example when CDATA is unnecessary:

```
<Text textformat="05"><![CDATA[<p>A literary thriller and a searing portrait of a
community riven by hatred.</p>]]></Text>
```

In this case, the markup is XHTML (textformat="05"), and XHTML is correct without CDATA. It's usually only HTML (textformat="02") that requires CDATA.

8. "Horrendous" HTML tagging

All the examples here are based on real ONIX, and the example above was actually this:

```
<Text textformat="05"><![CDATA[<p><p>A literary thriller and a searing portrait of a
community riven by hatred.</p></p>]]></Text>
```

Here's another:

```
<Text textformat="02">&lt;DIV&gt;&lt;FONT face=&amp;quot;Times New
Roman&amp;quot;&gt;&lt;b&gt;Christopher Owen&lt;/b&gt; holds degrees in modern
languages and history, and is the author of several anthologies of Occitan and
Basque oral poetry. He lives in France and Spain.&lt;/FONT&gt;&lt;/DIV&gt;</Text>
```

In a narrow technical sense, this second one's actually valid. But <DIV> and <FONT> are not part of the set of tags recommended for use in ONIX. Worse, use of FONT has been discouraged for years in HTML, and worst of all, it isn't valid at all in HTML 5. These and other unusual tags often get added if you cut and paste text from other sources into your ONIX.

It's best for recipients if senders stick to a relatively simple set of recommended tags:

paragraphs and line breaks	`<p>`	`<br />`			
emphasis, or book titles	`<strong>`	`<em>`	`<b>`	`<i>`	`<cite>`
bulleted and numbered lists	`<ul>`	`<ol>`	`<li>`		
sub- and superscript	`<sub>`	`<sup>`			
definition lists	`<dl>`	`<dt>`	`<dd>`		
simple glosses	`<ruby>`	`<rb>`	`<rp>`	`<rt>`	

And this – using XHTML without CDATA or escaping the markup – is so much simpler…

```
<Text textformat="05"><p><b>Christopher Owen</b> holds degrees in modern languages and
history, and is the author of several anthologies of Occitan and Basque oral poetry.
He lives in France and Spain.</p></Text>
```

An EDItEUR Application note on the correct use of embedded HTML and XHTML is available.

7. Trying to format text using spaces, tabs, returns

Something like this in the ONIX:

```
<Text textformat="06">
Foreword ................................ 1
Chapter 1 ............................... 5
Chapter 2 .............................. 31
Chapter 3 .............................. 47
…
</Text>
```

is fairly likely to end up being interpreted like this:

```
<Text textformat="06">Foreword ................................ 1 Chapter 1
............................... 5 Chapter 2 .............................. 31 Chapter
3 .............................. 47 …</Text>
```

Why? Because tabs, returns and multiple spaces in XML frequently get interpreted as single spaces – and even if they *are* correctly interpreted by the ONIX data recipient, when displayed on a web page the tabs, spaces and returns will be ignored (or more specifically, will be treated as a single space. As a data sender, if you want your text formatted as anything other than a single paragraph, the way to do it is to use XHTML markup.

6. Sending the wrong notification type

– for example always using notification type 03, sending 04 for a record that is NOT a partial or block update, or even sending 04 in ONIX 2.1 (in which it is not valid).

For recipients, <NotificationType> is mostly a rough guide to the reliability and stability of the metadata in the record. Type 01 indicates the data is subject to significant change (even key information like the binding, the title or author names) because it is sent a long time – usually several months – prior to publication, whereas most of the bibliographic metadata in a type 03 record should be correct and highly stable because Notification type 03 records should be sent no earlier than the publication date (or thereabouts – a few days earlier is fine). But Type 04 is completely different, in that recipients have to process block updates differently.

A special case here is Type 05 – a deletion. While its real meaning is 'please delete this record entirely, because it was issued in error', some ONIX data senders try to use type 05 to mean 'the product has been removed from the catalogue – you cannot purchase it any more'. Of course, upon receipt of such an 05 code, recipients often *cannot* delete the record, because they have customer orders or sales records linked to it. Senders should always use <PublishingStatus> to indicate 'the product has been removed from the catalogue' – either to indicate it is formally out of print or is unavailable indefinitely.

Similarly, some data senders use Type 05 when they abandon publication of a previously-planned product. Again, this is not what code 05 means, and abandonment too should be treated as a change of <PublishingStatus>.

5. Use of codes exclusive to a different version of ONIX

More generally, recipients decry the use of codes that are exclusive to ONIX 2.1 in ONIX 3.0 messages, or *vice versa*. The codelists for ONIX 2.1 and 3.0 are almost the same – but they are *not* identical. A few codes are unique to one or another version, and sometimes entire codelists are exclusive. So while:

```
<ProductForm>DG</ProductForm>
```

is valid in ONIX 2.1, its near-equivalent in 3.0 is:

```
<ProductForm>ED</ProductForm>
```

The ONIX tag has not changed, but the relevant codelist has – from list 7 to list 150.

Allied to this is the use of 'proprietary' codes where there is no provision for them [1] – typically a publisher decides its products are so different from those of other producers that they unilaterally create codes of their own. But how do data recipients know what these codes mean? Always ensure the codes you use come from the right codelists – and if you find a need for new codes, consult your ONIX national group or EDItEUR itself. There's a regular process for adding new codes to codelists so they can be used with certainty by everyone.

As an aside, validating a message that contains such 'proprietary' codes with the ONIX XSD or RNG schemas reveals the invalid code. Validating with the DTD does not – so such messages can *appear* to be valid even when they are not.

4. Failing to ensure the <RecordReference> is unique and does not change

For recipients, the record reference might be the single most important field, as it's used to decide whether newly-received metadata is a new product, or is an update of a previous record. Using the same Record reference for two different products means they 'fight', with one product's metadata overwriting the data for the other each time there is an update. Conversely, changing the Record reference on a single product almost inevitably means the recipient ends up with *two* records in their system.

Senders should ensure <RecordReference> is both unique and persistent throughout the entire life of the product. Using the ISBN as a Record reference is common, but even that can occasionally cause a problem, for example when an ISBN is assigned in error and has to be changed. It can also cause issues when a recipient receives metadata on the same product from multiple sources. Adding something like a domain name to the ISBN, or even better to an unchanging row ID from a database, works well (`<RecordReference>com.globalbookinfo.onix.01734529</RecordReference>` is used in the sample record supplied with the ONIX 3.0 *Specification*).

3. Sending prices without an associated territory

or assuming that currency implies country. On the face of it, this seems reasonable:

[1] Of course, there are several places where there *is* proper provision for proprietary codes, and use of proprietary codes in the right context isn't a problem at all.

```
<Price>
       <PriceType>01</PriceType>
       <PriceAmount>7.95</PriceAmount>
       <CurrencyCode>USD</CurrencyCode>
</Price>
<Price>
       <PriceType>01</PriceType>
       <PriceAmount>10.95</PriceAmount>
       <CurrencyCode>CAD</CurrencyCode>
</Price>
```

So roughly $8 in the USA, and $11 in Canada. But since there is no <Territory> composite within each price to indicate where that price applies, it means both prices are applicable throughout the market (or, in the case of a single market, throughout the territory with positive sales rights). It means retailers ordering their stock copies from the publisher or distributor in fact have a *choice* of prices – Canadian retailers could choose to use the slightly less expensive US price as the basis for their transactions [2]. In most cases, this isn't what the publisher intends.

So specify a price like this:

```
<Price>
       <PriceType>01</PriceType>
       <PriceAmount>10.95</PriceAmount>
       <CurrencyCode>CAD</CurrencyCode>
       <Territory>
              <CountriesIncluded>CA</CountriesIncluded>
       </Territory>
</Price>
```

to make it clear the territory within which each price is valid. An Application note on pricing is available.

2. Using inappropriate or unrealistic age ranges

– for example suggesting that a childen's book might be suitable for ages 1–99:

```
<AudienceRange>
       <AudienceRangeQualifier>17</AudienceRangeQualifier> <!-- interest age -->
       <AudienceRangePrecision>03</AudienceRangePrecision> <!-- from -->
       <AudienceRangeValue>1</AudienceRangeValue>
       <AudienceRangePrecision>04</AudienceRangePrecision> <!-- to -->
       <AudienceRangeValue>99</AudienceRangeValue>
</AudienceRange>
```

More subtly, most children's books have relatively narrow interest age ranges (and they are narrowest for young children). A book of interest to an 8- or 9-year-old is unlikely to appeal greatly to a teenager of 14, and a picture book for a 4-year-old would be of little interest to an 8-year old. 'Children of all ages' really doesn't apply, so keep age ranges narrow. <AudienceRange> is about identifying the *core* of the audience or readership for the book.

[2] Though currency exchange fluctuation could switch this around so that US retailers choose the CAD price…

This is not to deny there are a very few books that genuinely appeal to both children and adults: Asterix, a book of (non-cryptic) crosswords, or an illustrated companion to a popular film franchise might all appeal to those from 10 upwards. For these so-called ‘family’ audiences, you should use the <Audience> composite, and use <AudienceRange> to set just a minimum age for children:

```
<Audience>
       <AudienceCodeType>01</AudienceCodeType>
       <AudienceCodeValue>01</AudienceCodeValue> <!-- general/trade (adult) -->
</Audience>
<Audience>
       <AudienceCodeType>01</AudienceCodeType>
       <AudienceCodeValue>03</AudienceCodeValue> <!-- young adult (teenage) -->
</Audience>
<Audience>
       <AudienceCodeType>01</AudienceCodeType>
       <AudienceCodeValue>02</AudienceCodeValue> <!-- children/juvenile -->
</Audience>
<AudienceRange>
       <AudienceRangeQualifier>17</AudienceRangeQualifier> <!-- interest age -->
       <AudienceRangePrecision>03</AudienceRangePrecision> <!-- from -->
       <AudienceRangeValue>10</AudienceRangeValue>
</AudienceRange>
```

But be cautious: genuine appeal to such a broad audience is rare.

1. And finally, sending ONIX that doesn’t validate using the XSD schema

All too many senders rely on DTD validation – or on no validation at all…. It’s true that DTDs were the original technology for validation of ONIX in the early ’00s, but much better XSD (and RNG) validation tools have existed for ONIX since 2006. DTDs check only the most basic XML syntax and ONIX tags, whereas XSD and RNG schemas validate codes from codelists as well.

Now if a sender is using off-the-shelf product management software to create their ONIX, the output is very likely to be valid, and probably only needs to be checked when installing a new IT system or update, or during the onboarding process when establishing a brand new data feed. But validation – with the ‘classic’ XSD, or even better, the relatively new ‘strict’ XSD – is an essential part of testing a data feed. And the proportion of software developers who do not routinely use the XSD – and the newest strict XSD – in their work is startling.

EDItEUR has created two Application notes containing a step-by-step tutorial on validation with DTD and XSD. There are separate versions for Mac and Windows, and they cover using low-cost and free software tools as well as full-featured XML development environments like oXygen and XML Spy.

A final caveat

No adherence to best practice, and no validation of any kind will protect you if the data itself isn’t correct! Strict validation will point out the error if you claim a publication date of Feb 30th, but a date of Feb 28th is ‘valid’ – even if the correct date is actually January 28th.

Graham Bell
EDItEUR
7th April 2020