DBL Metadata 2.3 Documentation

Mark Howe

Mar 21, 2019

CONTENTS:

CHAPTER

ONE

AN INTRODUCTION TO XML AND DBL METADATA

1.1 XML STYLE

1.1.1 Human Readability

XML, like SGML, was designed to be readable and editable by non-technical users. It is of course possible to produce utterly opaque XML (and some companies specialize in doing this), but, within the Bible publishing world, non-programming archivists routinely skim XML for content and, when necessary, modify it using a text editor.

XML values readability over conciseness (within reason). Tag names tend to use unabbreviated words. Perhaps because of Java's domination of the XML world, composed names tend to be camelCase.

1.1.2 Attributes vs Elements

Choices need to be made between

```
<parentElement>
    <childElement1>content</childElement1>
    <childElement2>content</childElement2>
</parentElement>
```

vs

```
<parentElement childAtt1="content" childAtt2="content"/>
```

or maybe

```
<parentElement>
    <child n="1">content</child>
    <child n="2">content</child>
</parentElement>
```

There are some fairly solid rules to follow, such as

- Do not use attributes where the value may eventually need to be structured
- Do not use attributes for anything that needs localizing (since some locales require markup, CDATA does not work with attributes, etc)
- Do not put long text strings in attributes (because it looks ugly and is hard to read and format)
- Do not put large numbers of attributes in any one element (because it looks ugly and is hard to read and format)
- Do not use attributes where order matters (because not all technologies preserve attribute order)

There are also lots of strong opinions and habits that make for lively arguments at XML conferences.

1.1.3 Document Order

This is a basic XML concept, respected by all W3C XML technologies, which requires XML processors to respect the order in which elements appear. So, eg, in

```
<parent>
    <child>foo</child>
    <child>baa</child>
    <child>frob</child>
</parent>
```

it will always be possible to discover that "baa" appears after "foo" and before "frob". In other words, XML does not have a clear distinction between lists/arrays and dictionaries/objects. Thus, regarding

```
<structure>
    <content name="book-gen" src="release/USX_2/GEN.usx" role="GEN"/>
    <content name="book-exo" src="release/USX_2/EXO.usx" role="EXO"/>
    <content name="book-lev" src="release/USX_2/LEV.usx" role="LEV"/>
    <content name="book-num" src="release/USX_2/NUM.usx" role="NUM"/>
    <content name="book-deu" src="release/USX_2/DEU.usx" role="DEU"/>
</structure>
```

XML technologies can iterate efficiently over the content elements, in the expected "document" order, or access one or more content element efficiently using any combination of attribute values. There is no need for tricks like "001_GEN" to preserve order.

(It is obviously possible to parse XML into an unordered dictionary, at which point the elements will be... unordered. Also, XML does not announce when order is considered to be significant, since significance is somewhat in the eye of the XML producer and/or consumer.)

1.2 PROCESSING XML

1.2.1 SAX

SAX is one of the oldest XML processing models that is supported by most languages. It is a streaming model that uses callbacks for every syntactic element, such as a start tag, an end tag, text...

SAX can be very fast - in a C implementation the limiting factor is often the bandwidth of the storage device. It is often used to bootstrap other models, and has become more popular recently on mobile devices because of its low memory usage. It can be an elegant way to "cherry pick" a few features the document while ignoring most of the content and/or structure.

However, many developers dislike SAX because

- there is no context for callbacks unless the application code saves it
- there is no way to look forward
- for complete processing of complex documents, the SAX callback handlers tend to become nested conditionals that are hard to debug and potentially fragile when faced with unexpected document content.

1.2.2 DOM

DOM is the original XML tree model, and is probably one of the most popular ways for applications to process XML. (Like SAX, DOM is a model rather than a standard, so implementational details vary.) Basic DOM operations allow typical tree manipulation functionality such as

- find parent, children
- remove, add or move a node
- discover the type of a node
- count nodes

The main challenge with DOM is that it does little to hide the complexity of XML documents. For example, changing the whitespace between elements in a document can drastically change the DOM representation of that document.

1.2.3 XPath

XPath may be viewed as a way to describe a route to find specific parts of a document. It consists of one or more step, separated by slashes, eg

/DBLMetadata/identification/systemId[@type='gbc']

which can be read as

- Start with the root element which is "DBLMetadata"
- Get all the child elements with a tag of "identification" (in practice there's only one of those)
- Get all the child elements with a tag of "systemId" with an attribute called "type" that has a value of "gbc"

The result is a list of zero or more nodes which can then be processed by various technologies.

Most modern implementations of DOM include XPath 1.0, as do some DBMSs such as PostGreSQL. XPath 2.0 is more powerful and consistent, but is not well-supported, especially in the non-JVM open-source world.

1.2.4 XQuery

XQuery is a superset of XPath 2.0 which provides something like SQL functionality for working with one or more XML document.

1.2.5 XSLT

XSLT is a turing-complete, functional XML vocabulary for transforming XML documents. The basic unit of an XSLT stylesheet is the template. Parsing can be directed explicitly, much as with XQuery, or templates can be matched as part of all of the document is traversed. XSLT is particularly useful for making a limited number of changes to a complex document, most of which should be copied. (This is based on an identity transform.) XSLT tends to polarize developers, between those who consider it to be Lisp with pointy parentheses and those who consider it to be arcane and verbose.

XSLT 1.0 is available for most languages. XSLT 2.0 and 3.0 are not well-supported in the open-source world.

1.2.6 Other Processing Models

There are many language-specific processing models. Most of these models attempt to make XML simpler, or more like something else such as nested objects or a database. This tends to work well for the simple cases and not at all for the hard ones. (Scala is one example, where namespace-broken XML support is baked into the core language.)

Regular expressions are the XML processing model that no-one admits to using, but that most people end up using at some point. It can work remarkably well (I've seen conversion of OSIS to USX using cascading regexes in PHP), but it tends to be fragile and struggles with recursive structures. Regexes are one example of reinventing the XML parser, which is generally considered to be A Bad Idea (since handling all the XML edge cases is remarkably hard, and since the whole point of XML is to provide a consistent syntax so that application code doesn't need to worry about character-based parsing at all.)

1.3 XML VALIDATION

1.3.1 Validity vs Well-Formedness

This is an important distinction in XML, which is less clear with other document formats (perhaps because XML has unusually good validation support.)

Well-formed means that the document is XML, eg the tags match and are correctly constructed. Most XML processors will stop dead at the first sign that the document is not well-formed.

Valid* means that the document conforms to a particular schema.

There are therefore three levels of correctness:

- Not well-formed, ie it isn't XML at all
- Invalid, ie it's XML but not the XML we were expecting
- Valid, ie it's the XML we were expecting

1.3.2 The case for strict schema

Validation errors can be frustrating and, in some cases, hard to pinpoint. However, strict validation also has major benefits, including

- the schema provides a machine-executable, formal definition of the "shape" of a document. This is hugely preferable to a verbose description in a human language, which is inevitably ambiguous and which then needs to be implemented anyway.
- validated documents can be processed with very little defensive code, because there are no surprises on the level of missing data, unexpected data or data of an unexpected type. This leads to shorter, cleaner, more maintainable programs.

(The corollary of this is that *Bad Things Will Happen* if systems built to assume valid documents receive invalid documents and do not perform their own validation.)

1.3.3 DTDs

This was the first attempt to describe the structure of XML documents. DTDs are still used, but have generally been replaced by schemas, which describe XML using XML (or something that can be losslessly converted into XML).

1.3.4 XML Schema (XSD)

This is the original W3C schema specification. v1.0 is widely implemented. It is powerful, but has been criticised for its sprawling spec and its inability to validate some document features. v1.1 was intended to address these concerns, and can now validate everything that any other schema language can validate... but is even more sprawling. Open-source support for v1.1 is patchy.

1.3.5 RelaxNG Schema (RNG/RNC)

This was an attempt to provide an alternative to XSD. It is generally agreed that RelaxNG schema are easier to write (and to read!) than XSD, especially when the schema is expressed in the "compact" syntax that bears a passing resemblance to Bachus Naur notation. In addition, RelaxNG can validate documents that cannot be parsed statically, typically because the permitted high-level structure depends on the low-level structure. DBL Metadata is one such document, which is one reason why the current schema is written in RelaxNG.

RelaxNG 1.0 is well-supported in the open source community. One drawback, which is a corollary of dynamic parsing, is that error messages are not always very informative.

1.3.6 Schematron

In contrast to XSD and RelaxNG schema, schematron can be considered to be a way to write unit tests for XML documents. Schematron would be a very clumsy way to validate every aspect of a document. However, it is particularly useful for checking consistency between parts of a document, or for detecting duplicate values. Various versions of Schematron are available, with reasonable support for Schematron 1.6 via libxml2.

1.4 DBL METADATA CONVENTIONS

1.4.1 Dublin Core

DBL Metadata was inspired by Dublin Core, a set of standard metadata names. (DC does not provide everything needed for our domain, and in some cases the DC approach seemed overly clumsy for our purposes.)

1.4.2 Elements vs Attributes

DBL Metadata generally uses child elements for most content, reserving attributes for qualifiers and machine-readable keys, eg

```
<systemId type="ptreg">
    <id>kK6ASA9ScumywfT9v</id>
</systemId>
```

1.4.3 Order

In most places within DBL Metadata, order is unimportant. (One exception is publication structure which describes something like a contents page.) DBL tends to stick to a well-known order of top-level elements, and sometimes rearranges documents to follow this order, but this is purely to make eyeballing easier, and no technology should rely on this order.

1.4.4 Optional, non-empty elements

As of v2.0, the DBL Metadata schema has many optional elements and few elements that may be empty. So, for example, in

```
<identification>
    <name>Malayalam Bible [mal] India (BCS 2017)</name>
    <nameLocal>&#3374;&#3378;&#3375;&#3390;&#3379;&#3330; &#3372;&#3400;&#3372;&#3391;
↪&#3379;&#3405;&#8205;</nameLocal>
    <description>The Holy Bible in the Malayalam language of India (BCS 2017)</
↪description>
</identification>
```

- name and description are required
- nameLocal is optional but present
- descriptionLocal is optional and not present (but never present and empty)

```
<identification>
    <name>Malayalam Bible [mal] India (BCS 2017)</name>
    <nameLocal/>
    <description>The Holy Bible in the Malayalam language of India (BCS 2017)</
↪description>
    <descriptionLocal></descriptionLocal>
</identification>
```

is invalid because two elements are empty. (The example shows two equivalent ways of writing an empty element in XML.) There are two arguments for this approach:

- in some cases there is a semantic difference between “no value” and “a value of ‘’”
- it is not uncommon for bugs such as incorrect xpaths to result in empty elements, and it is good to detect such errors.

1.4.5 Inheritance

Some information may be specified at several levels. For example, the name of entry could be obtained from (in order)

- the nameLocal element of the selected publication
- the name element of the selected publication
- the nameLocal element in the identification section
- the name element of the identification section

In the interests of consistency, the optional elements should only be provided where the value differs from a more general value. So, eg there is no need to provide a name for a publication if the required name (or nameLocal) is identical to the name in the identification section. This is one of the less popular design decisions, but it seems important to avoid copy-and-paste fixing of empty fields, eg

```
<identification>
    <name>Malayalam Bible [mal] India (BCS 2017)</name>
    <nameLocal>Malayalam Bible [mal] India (BCS 2017)</nameLocal>
</identification>
```

where there is no easy way to decide if nameLocal is a placeholder value or the actual desired value. The correct way to represent this would be

```
<identification>
    <name>Malayalam Bible [mal] India (BCS 2017)</name>
</identification>
```

and, ideally, a nameLocal element with a localized value would be added at some point.

1.4.6 Schema

DBL Metadata is currently validated using a RelaxNG schema for structure plus a Schematron schema for constraints (mainly checking for hanging references).

CHAPTER

TWO

IDS

2.1 IN DBL METADATA 2.2

2.1.1 Entry ID

16-hex string derived (for text entries) from the Mercurial ID.

2.1.2 User ID

Not currently exposed in the metadata.

2.1.3 Agency ID

16-hex string.

2.1.4 License ID

Not currently exposed in the metadata.

2.2 PROPOSED CHANGES FOR 2.3

2.2.1 Optional idServer declarations

When present, these would appear as the first children of the root element, eg

```
<idServer prefix="dbl">https://thedigitalbiblelibrary.org</idServer>
<idServer default="true">http://atlantisbibleconsortium.net</idServer>
<idServer prefix="myServer" local="true">http://localhost::8080</idServer>
```

- Element name is “idServer”
- The element must contain either a ‘prefix’ or a ‘default=”true”’ attribute. (It may contain both, and only one idServer may be the default.) The default, if present, will be assumed to apply to namespaces with no prefix.
- The element may contain a ‘local’ attribute. When true, this signifies that the ids are for internal use only, and that they should be stripped before export.

- The enclosed text is a URI. In schema this means "pretty much any string", but using URLs that resolve to a server endpoint would help with discoverability.

If there is no default idServer, all ids in the document must be prefixed.

If there are no idServer declarations, no ids may be prefixed and all ids are assumed to refer to DBL (as in v2.2).

2.2.2 ID Syntax

This is surprisingly challenging since

- we need to allow for a wide range of ids from diverse systems
- we need to be able to distinguish prefixes from the start of an unqualified idServer
- it is considered a bad id to use XML namespace-like notation for things that are not XML namespaces (ie no single colons)

The proposed solution is

(<prefix>::)?<id>

where

- "prefix" is a NCName (an XML name with no colon)
- "id" matches

```
[0-9A-Za-z]([0-9A-Za-z_-]{0,30}[0-9A-Za-z])?
```

ie a string starting and ending with an alphanumeric character and containing alphanumeric characters, hyphens and underscores.

IDs in this format can be tested for prefixedness (?!) by searching for "::", which seems unlikely to occur in any existing id schemes.

2.2.3 Revision Syntax

The non-prefixed ID regex above ought to allow DBL (numeric) and PT/uW (UUID) revision/commit identifiers.

2.2.4 Expose User ID

This should happen anywhere that the metadata refers to a person by name. It probably needs to be optional since it may not be possible to recover this information retrospectively.

2.2.5 Expose License ids

This would be part of the new license subsection.

2.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

- Remove DBL-as-default behavior (which means, among other things, that at least one idServer element would be required).

CHAPTER

THREE

THE ROOT ELEMENT

3.1 IN DBL METADATA 2.2

```
<DBLMetadata version="2.2" id="9f78f34aabe691c9" revision="3">
```

- The root element name is DBLMetadata in the null namespace (ie no namespace is declared for the root element).
- @version is required and can be 2.1, 2.1.1 or 2.2.
- @id (required) is the DBL entry id (16 chars hex)
- @revision (required) is the revision number (a positive integer)

3.2 PROPOSED CHANGES FOR 2.3

- @version must be 2.3 (since there are breaking changes)
- @id regex should be expanded to allow optional prefixes and other formats of id. (We can do this so that unqualified ids must match DBL's strict regex)
- @revision should be expanded to allow Mercurial and Git commit ids. (We can do this so that unqualified ids must match DBL's strict regex)

3.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

3.3.1 Root element name

The name should have something to do with Scripture Burrito, and we should probably state that it's for metadata since, sooner or later, there will be other Scripture Burrito schema.

The name should remain in the null namespace to lower processing barriers for what the XML spec describes as "the desperate Perl programmer". so

- SBMetadata?
- ScriptureMetadata?
- ScriptureBurritoMetadata?

3.3.2 @revision ==> @commit?

Maybe not, if "revision" is a more neutral term than "commit".

3.3.3 @id and @revision in root element?

In Scripture Burrito we may have multiple ids for a snapshot, and DBL may not be in any sense the primary id. There are at least two ways to represent this:

- One id/revision can be placed in the root element, with the others in systemId
- No ids/revisions are placed in the root element, with all such data in systemId

The second option is more orthogonal. However, having basic identification info at the top of the document makes eyeballing easy and is also extremely convenient for streaming processors such as SAX. (Briefly, it's nice to know early on what you are processing, where you might store it, etc.) The first option would imply multiple ways to represent the same set of ids, with the possibility of rotating any systemId into the root element.

CHAPTER

FOUR

IDENTIFICATION

4.1 TOP-LEVEL FIELDS

4.1.1 IN DBL METADATA 2.2

/DBLMetadata/identification (Exactly 1)

- **name** (Exactly 1 string)
  - The entry's name, in English
    - regex: S.*S
- **nameLocal** (0 or 1 string)
  - The entry's localized name
    - regex: S.*S
- **abbreviation** (Exactly 1 string)
  - The entry's abbreviation, in English (no exotic characters)
    - regex: [-A-Za-z0-9]{2,12}
- **abbreviationLocal** (0 or 1 string)
  - The entry's localized abbreviation
    - regex: S.{0,10}S
- **description** (Exactly 1 string)
  - The entry's description, in English
    - regex: S.*S
- **descriptionLocal** (0 or 1 string)
  - The entry's localized description
    - regex: S.*S
- **scope** (Exactly 1 string)
  - The entry's scope (across all publications)
    - Enum:
      - Bible
      - Bible with Deuterocanon

        - New Testament
        - New Testament+
        - Old Testament
        - Old Testament + Deuterocanon
        - Old Testament+
        - Portions
        - Selections
        - Shorter Bible
- **dateCompleted** (0 or 1 string)
  - The date on which this entry was completed
    - regex: [12]d{3}(-[01]d(-[0-3]d(T[012]d:[0-5]d:[0-5]d)?)?)?
- **bundleProducer** (Exactly 1 string)
  - The client and client version that produced this bundle
    - regex: S.*S

4.1.2 PROPOSED CHANGES FOR 2.3

4.1.2.1 Multiple *Local elements

- nameLocal
- descriptionLocal
- abbreviationLocal

4.1.2.2 basedOn

This would uniquely identify the snapshot on which the entry is based, which might be from a different ecosystem. This information is potentially useful for forensics. It also provides a mechanism for 3-way diffing of documents when the two deltas are from different ecosystems.

```
<basedOn type="dbl">
   <id>482ddd53705278cc</id>
   <revision>1</revision>
</basedOn>
```

4.1.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

4.1.3.1 Evaluate uses of scope

Scope does not have enough options to describle all projects. In addition, it is unclear whether the scope describes the books actually present (impossible with an enum for incremental publishing) or the intended final scope of the project (which is a somewhat existential concept). Something like the canonicalContent section in publications, for a whole entry, would provide scope information in a more flexible and transparent way.

4.2 systemId

4.2.1 IN DBL METADATA 2.2

/DBLMetadata/identification/systemId (Exactly 1)

- **/DBLMetadata/identification/systemId[@type='gbc']** (0 or 1)
  - **id** (Exactly 1 string)
    - The GBC id (24 hex characters)
      - regex: [0-9a-f]{24}
- **/DBLMetadata/identification/systemId[@type='paratext']** (0 or 1)
  - **id** (Exactly 1 string)
    - The paratext id for this entry (40 hex characters)
      - regex: [0-9a-f]{40}
  - **name** (Exactly 1 string)
    - The Name for this ID
      - regex: S.*S
  - **fullName** (Exactly 1 string)
    - The Full Name for this ID
      - regex: S.*S
  - **csetId** (0 or 1 string)
    - The CSet id for this ID
      - regex: S.*S
- **/DBLMetadata/identification/systemId[@type='ptreg']** (0 or 1)
  - **id** (Exactly 1 string)
    - The Paratext Repository id (17 hex characters)
      - regex: [0-9a-zA-Z]{17}
- **/DBLMetadata/identification/systemId[@type='tms']** (0 or 1)
  - **id** (Exactly 1 string)
    - The TMS id for this entry (an UUID)
      - regex: [0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}
- **/DBLMetadata/identification/systemId[@type='reap']** (0 or 1)
  - **id** (Exactly 1 string)
    - The REAP id for this entry (an UUID)
      - regex: [^]+
- **/DBLMetadata/identification/systemId[@type='biblica']** (0 or 1)
  - **id** (Exactly 1 integer)
    - The Biblica ID (a number)

            - max: 99999

- **/DBLMetadata/identification/systemId[@type='dbp']** (0 or 1)
  - **id** (Exactly 1 string)
    - The DBP id for this entry (10 hex characters)
      - regex: [A-Z0-9]{10}

4.2.2 PROPOSED CHANGES FOR 2.3

4.2.2.1 Add DBL type

This is required to make the document structure orthogonal.

4.2.2.2 Add other known organizations

- Unfolding Word
- Vachan Online

4.2.2.3 Support x-* types

The systemId type mechanism was created when DBL needed to work with a small number of large ecosystems. Future ecosystems may be small – maybe a national denomination or even one church. It may not always make sense to add such organizations to the schema and, when it does, this will take some time. Some architectures involve local servers (on a VPN, an intranet or even localhost), and testing sometimes requires server changes. Supporting types matching

provides a way to introduce new or private ecosystems without rewriting schema:

```
<idServer prefix="mvah">https://markspersonaltranslationproject.fr</idServer>
...
<systemId type="x-mvah">
   <id>idInMyPersonalFormat</id>
   <myDetail>something-that-interests-me</myDetail>
</systemId>
```

The type of all x-* systemIds should correspond to an idServer declaration.

4.2.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

None.

4.3 canonSpec

4.3.1 IN DBL METADATA 2.2

This feature was added as a more flexible and transparent alternative to the scope and tradition values, and as a first step towards hierarchical publication structures. It is based on analysis of the Canons.xml used by Paratext. It is currently not used by Paratext, but is central to Nathanael's workflow.

/DBLMetadata/identification/canonSpec (0 or 1)

- **@type** (Exactly 1 string)
  - The overall structure and order of this canon. (OT+ here means canonical and deuterocanonical OT books interleaved within the same section, like most Catholic Bibles)
    - Enum:
      - OT
      - OT+
      - DC
      - NT
      - OT, NT
      - OT+, NT
      - OT, NT, DC
      - OT, DC, NT
- **component** (1 or more string)
  - The components of this canon, which should match the canon type chosen above. eg, if the canon type is "OT, NT", there should be one OT and one NT component here.
    - Enum:
      - armenianApostolicDC
      - armenianApostolicOT
      - armenianApostolicOT2
      - armenianClassicalOT
      - armenianNT
      - catholicAndAnglicanDC
      - catholicLxxDC
      - catholicLxxOT
      - catholicLxxSeparatedDC
      - catholicPlusLutheranDC
      - catholicVulgateDC
      - catholicVulgateOT
      - catholicVulgateSeparatedDC
      - czechKralickaDC
      - danishLutheranDC
      - ethiopianOrthodoxDC
      - ethiopianOrthodoxNT
      - ethiopianOrthodoxOT
      - ethiopianProtestantNT
      - ethiopianProtestantOT

- georgianOrthodoxDC
- georgianOrthodoxOT
- georgianOrthodoxOT2
- georgianSynodalDC
- germanLutheranDC
- greekOrthodoxDC
- greekOrthodoxOT
- kjvDC
- kjvNonDC
- lutheranNT
- romanianOrthodoxDC
- romanianOrthodoxOT
- russianNT
- russianOrthodoxDC
- russianOrthodoxOT
- russianProtestantOT
- russianSynodalDC
- syriacNT
- syriacOT
- tanakhOT
- turkishInterconfessionalDC
- vulgateCatholicBible
- westernInterconfessionalDC
- westernInterconfessionalDC2
- westernNT
- westernOT

4.3.2 PROPOSED CHANGES FOR 2.3

4.3.2.1 Remove the .*2 components

These variants of three components correspond to longstanding inconsistencies in the Canons.xml file, caused by inconsistent use of DAN/DAG and EST/ESG in canons of different scope for the same tradition (eg the OT part of the Armenian Bible canon does not match the Armenian OT canon). Also, there is no JER in the Greek Orthodox canon.

4.3.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

4.3.3.1 Develop canonSpec

One day, canonSpecs should be able to use custom components, which begs the question of where and how those components would be defined.

CHAPTER

FIVE

RELATIONSHIPS

5.1 IN DBL METADATA 2.2

/DBLMetadata/relationships (Exactly 1)

/DBLMetadata/relationships/relation (0 or more)

- **@id** (Exactly 1 string key)
  - The DBL id of the related entry
    - regex: [a-f0-9]{16}
- **@revision** (Exactly 1 integer)
  - The revision of the related entry
    - min: 1
- **@relationType** (Exactly 1 string)
  - The role of the related entry with respect to this entry
    - Enum:
      - source
      - expression
- **@type** (Exactly 1 string)
  - The medium of the related entry
    - Enum:
      - text
      - audio
      - print
      - video
      - braille
- **@publicationId** (0 or 1 string)
  - The publication in the related text entry on which to base the braille
  - regex:
    - `[A-Za-z][A-Za-z0-9_\-]{0,31}`

5.2 PROPOSED CHANGES FOR 2.3

5.2.1 Expand @relationType enum

This mechanism could be used to represent other entry-to-entry relationships, eg between Bible text and related para-biblical material.

5.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

There has been some discussion about resource-to-resource relationships. The relationships section probably isn't the best place to address this.

CHAPTER

SIX

AGENCIES

6.1 IN DBL METADATA 2.2

/DBLMetadata/agencies (Exactly 1)

- **/DBLMetadata/agencies/rightsHolder** (1 or more)
  - **uid** (Exactly 1 string key)
    - The id of this rights holder
      - regex: [a-f0-9]{24}
  - **name** (Exactly 1 string)
    - The name in English of this rights holder
      - regex: S.*S
  - **nameLocal** (0 or 1 string)
    - The local name of this rights holder
      - regex: S.*S
  - **abbr** (Exactly 1 string)
    - The abbreviation of this rights holder
      - regex: S.*S
  - **url** (0 or 1 string)
    - The URL of this rights holder
      - regex: S.*S
- **/DBLMetadata/agencies/contributor** (1 or more)
  - **uid** (Exactly 1 string key)
    - The id of this contributor
      - regex: [a-f0-9]{24}
  - **name** (Exactly 1 string)
    - The name of this contributor
      - regex: S.*S
  - **content** (Exactly 1 boolean)
    - Contributes to Content?

  - **finance** (0 or 1 boolean)
    - Contributes to Finance?
  - **management** (0 or 1 boolean)
    - Contributes to Management?
  - **qa** (0 or 1 boolean)
    - Contributes to Quality Assurance?
  - **publication** (0 or 1 boolean)
    - Contributes to publication?
- **/DBLMetadata/agencies/rightsAdmin** (0 or 1)
  - **uid** (Exactly 1 string key)
    - The id of this rights administrator (24 chars of hex)
      - regex: [a-f0-9]{24}
  - **name** (Exactly 1 string)
    - The name of this rights administrator
      - regex: S.*S
  - **url** (0 or 1 string)
    - The URL of this rights administrator
      - regex: S.*S

6.2 PROPOSED CHANGES FOR 2.3

6.2.1 Multiple *Local elements

- allow multiple instances of nameLocal? (Currently no nameLocal.)

6.2.2 Less compulsory fields for upload variant

Some background... DBL Metadata is used as the basis of the job spec at the heart of uploading. In this variant more values are optional - notably @revision since this will be overwritten by the server in any case.

Right now, all the denormalized fields in the agencies section are required. However, the uids in the agencies section may only be changed via the DBL website, ie revisions attempting to change ownership will be rejected. This means that clients need to generate a lot of boilerplate, some of which has to be identical to the information on the server, and some of which is not validated for coherence by the server. So, eg, it is currently possible to change the name and url of a rightsHolder but not the uid, which is a recipe for utter confusion.

The proposal is to make many or maybe all fields optional when revision > 1, to reduce boilerplate and to not create false expectations about what may be changed via the client.

6.2.3 contributor/content should be optional

This is the only role-type field that is required for contributors, and was probably a typo in the schema.

6.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

Proper support for public licenses may have implications here.

CHAPTER

SEVEN

LANGUAGES

7.1 IN DBL METADATA 2.2

- **iso** (Exactly 1 string)
  - The language's 3-character ISO 639-3 code
    - regex: [a-z][a-z][a-z]
- **name** (Exactly 1 string)
  - The name, in English, of the language
    - regex: S.*S
- **nameLocal** (0 or 1 string)
  - The localized name of the language
    - regex: S.*S
- **scriptCode** (Exactly 1 string)
  - The ISO 15924 script code used in this entry
    - Enum:
    - Adlm
    - Afak
    - Aghb
    - Ahom
    - Arab
    - Aran
    - Armi
    - Armn
    - Avst
    - Bali
    - Bamu
    - Bass
    - Batk
    - Beng

* Blis
* Bopo
* Brah
* Brai
* Bugi
* Buhd
* Cakm
* Cans
* Cari
* Cham
* Cher
* Cirt
* Copt
* Cprt
* Cyrl
* Cyrs
* Deva
* Dsrt
* Dupl
* Egyd
* Egyh
* Egyp
* Elba
* Ethi
* Geok
* Geor
* Glag
* Goth
* Gran
* Grek
* Gujr
* Guru
* Hang
* Hani
* Hano
* Hans

* Hant
* Hatr
* Hebr
* Hira
* Hluw
* Hmng
* Hrkt
* Hung
* Inds
* Ital
* Java
* Jpan
* Jurc
* Kali
* Kana
* Khar
* Khmr
* Khoj
* Kitl
* Kits
* Knda
* Kore
* Kpel
* Kthi
* Lana
* Laoo
* Latn
* Latf
* Latg
* Lepc
* Limb
* Lina
* Linb
* Lisu
* Loma
* Lyci

* Lydi
* Mahj
* Mand
* Mani
* Marc
* Maya
* Mend
* Merc
* Mero
* Mlym
* Modi
* Mong
* Moon
* Mroo
* Mtei
* Mult
* Mymr
* Narb
* Nbat
* Nkgb
* Nkoo
* Nshu
* Ogam
* Olck
* Orkh
* Orya
* Osge
* Osma
* Palm
* Pauc
* Perm
* Phag
* Phli
* Phlp
* Phlv
* Phnx

* Plrd
* Prti
* Rjng
* Roro
* Runr
* Samr
* Sara
* Sarb
* Saur
* Sgnw
* Shaw
* Shrd
* Sidd
* Sind
* Sinh
* Sora
* Sund
* Sylo
* Syrc
* Syre
* Syrj
* Syrn
* Tagb
* Takr
* Tale
* Talu
* Taml
* Tang
* Tavt
* Telu
* Teng
* Tfng
* Tglg
* Thaa
* Thai
* Tibt

        - Tirh
        - Ugar
        - Vaii
        - Visp
        - Wara
        - Wole
        - Xpeo
        - Xsux
        - Yiii
        - Zinh
        - Zmth
        - Zsym
        - Zxxx
        - Zyyy
        - Zzzz
- **script** (Exactly 1 string)
    - The name of the script used in this entry
        - regex: S.*S
- **scriptDirection** (Exactly 1 string)
    - The direction of the script used in this entry
        - Enum:
        - LTR
        - RTL
- **numerals** (0 or 1 string)
    - The numerals system used in this entry
        - Enum:
        - Arabic
        - Bengali
        - Burmese
        - Chinese
        - Cyrillic
        - Devanagari
        - Ethiopic
        - Farsi
        - Gujarati
        - Gurmukhi

        * Hebrew
        * Hindi
        * Kannada
        * Khmer
        * Malayalam
        * Oriya
        * Roman
        * Tamil
        * Telugu
        * Thai
        * Tibetan
- **ldml** (0 or 1 string)
    - The LDML of the language
        * regex: [A-Za-z]{2,3}([-_][A-Za-z0-9]+){0,4}
- **rod** (0 or 1 string)
    - The ROD of the language
        * regex: [0-9]{5}

7.2 PROPOSED CHANGES FOR 2.3

7.2.1 BCP 47

This is the currently preferred way to store language information, as it can include most other standards and can handle minority languages and dialects that are unlikely to be supported by the older standards such as ISO 639-3.

It seems that the field currently called "ldml" is actually closer to BCP 47. This should be renamed and we should check that it handles all BCP 47 permutations.

We also need to decide whether to store the components of BCP 47 separately, since decomposing BCP 47 is not trivial. (For example, there is no 639-3 code if a 639-1 code exists.) At that point, another option would be to only store the components, which reduces duplication but pushes the onus of constructing BCP 47 onto the consumer.

7.2.2 Multiple Language Support

In the current schema there is exactly one language element. To support multiple languages we should adopt a structure similar to the current countries structure, ie a "languages" wrapper with one or more "language" child element. Exactly one of those languages should be marked as the default.

Here, as elsewhere, name and nameLocal need rethinking to support multiple languages. In addition to languages of the scriptural content, there may be localization languages. So, eg, an English-French diglot might contain localization strings in Spanish, Arabic or Hindi in order to localize interfaces.

In a world where English is no longer always the default language, it might make sense to abandon the name/nameLocal distinction altogether and, instead, to require one or more name, each in a specified language. The

downside of this is that every consumer then needs to implement some form of language negotiation to handle the case where, say, the preferred language is French and the options in the metadata are Chinese and Swahili.

7.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

None.

CHAPTER

EIGHT

COUNTRIES

8.1 IN DBL METADATA 2.2

- **/DBLMetadata/countries/country** (1 or more)
  - **iso** (Exactly 1 string key)
    - The country's 2-character country code
    - regex: [A-Z][A-Z]
  - **name** (Exactly 1 string)
    - The country's name in English
    - regex: S.*S
  - **nameLocal** (0 or 1 string)
    - The country's localized name
    - regex: S.*S

8.2 PROPOSED CHANGES FOR 2.3

8.2.1 Multiple Language Support

See the languages section.

8.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

None.

CHAPTER

NINE

NAMES

9.1 IN DBL METADATA 2.2

- **/DBLMetadata/names/name** (0 or more)
  - **@id** (Exactly 1 string key)
    - The id of this name
    - regex: [A-Za-z][-A-Za-z0-9_]+
  - **short** (Exactly 1 string)
    - The short label for this name, which is required and will be used as a default for the other labels if necessary
    - regex: S.(.{0,253}S)?
  - **abbr** (0 or 1 string)
    - The abbreviation for this name
    - regex: S(.{0,253}S)?
  - **long** (0 or 1 string)
    - The long label for this name
    - regex: S(.{0,1022}S)?

9.2 PROPOSED CHANGES FOR 2.3

This might mean allowing multiple abbr, short and long, distinguished by language attribute.

9.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

None

CHAPTER

TEN

COPYRIGHT

10.1 IN DBL METADATA 2.2

This section can contain long and/or short versions of the copyright statement, either of which may be in plain text and/or xhtml. (Historically, most DBL entries have the long version in XHTML.)

/DBLMetadata/copyright (Exactly 1)

- **/DBLMetadata/copyright/fullStatement** (0 or 1)
  - **statementContent[@type='xhtml']/*** (0 or 1 xml)
    - The copyright statement in DBL's subset of XHTML (must be valid XML, ie tags must match.)
      - schema: dbl/2_2/dbl-xhtml
  - **statementContent[@type='plain']** (0 or 1 string)
    - The copyright statement in plain text

/DBLMetadata/copyright/shortStatement (0 or 1)

- **statementContent[@type='xhtml']** (0 or 1 xml)
  - The copyright statement in DBL's subset of XHTML (must be valid XML, ie tags must match.)
    - schema: dbl-xhtml
- **statementContent[@type='plain']** (0 or 1 string)
  - The copyright statement in plain text

10.2 PROPOSED CHANGES FOR 2.3

10.2.1 Language attribute for statementContent

To support multiple languages, we would need to add a language attribute (or subelement) to each statement. This could be optional if there is only one language in the languages section.

10.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

Proper support for public licenses may have implications here.

CHAPTER

ELEVEN

PROMOTION

11.1 IN DBL METADATA 2.2

/DBLMetadata/promotion (Exactly 1)

- **/DBLMetadata/promotion/promoVersionInfo** (0 or 1)
  - **./*** (0 or 1 xml)
    - Promotional material in DBL's subset of XHTML (must be valid XML, ie tags must match.)
      - schema: dbl/2_2/dbl-xhtml

11.2 PROPOSED CHANGES FOR 2.3

11.2.1 Replace promoVersionInfo with statementContent

Right now the promotion section is similar to but confusingly different to the copyright section. A more coherent structure that also allows plain text promotional material would be

/DBLMetadata/promotion (Exactly 1)

- **statementContent[@type='xhtml']/*** (0 or 1 xml)
  - Promotional material in DBL's subset of XHTML (must be valid XML, ie tags must match.)
  - schema: dbl/2_2/dbl-xhtml
- **statementContent[@type='plain']** (0 or 1 string)
  - Promotional material in plain text

11.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

None.

CHAPTER

TWELVE

PROGRESS

12.1 IN DBL METADATA 2.2

DBL Metadata currently supports two mechanisms for tracking translation progress. The first, which is supported by PT, uses a top-level section to list the progress for each book:

```
<progress>
    <book code="GEN" stage="4"/>
    <book code="EXO" stage="1"/>
    <book code="JOS" stage="2"/>
    <book code="LUK" stage="4"/>
</progress>
```

There are two issues with this:

- It means another potentially long list in the metadata
- More importantly, it can only record progress for books when, in reality, progress on introductions and other para-canonical content may also be important.

The alternative mechanism, which is defined in the schema, but which has probably never been used, is to record the progress against manifest entries.

```
<resource checksum="0e6c24ebcf1ca2e928578ab239b69687" mimeType="application/xml" size=
↪"296803" uri="release/USX_2/1CH.usx" progress=”37”/>
```

Progress can therefore be logged against any document in the entry, without bloating the metadata document. One possible argument against this approach is that project tracking and manifest information may be generated by very different routes. Also, PT currently duplicates most canonical content several times when multiple booklists are specified (but maybe we should fix the duplication of content).

12.2 PROPOSED CHANGES FOR 2.3

We should pick one of these options, or come up with a new one, ensure that it will be supported by Paratext, and remove the unused options.

12.3 ISSUES TO CONSIDER FOR SCRIPTURE BURRITO

None.