**Alternative implementation**

STEPBible



**Contents**

[1 Introduction 3](#_Toc152749531)

[2 Terminology 3](#_Toc152749532)

[3 Issues with the present implementation 3](#_Toc152749533)

[3.1 Paragraph-level formatting 3](#_Toc152749534)

[3.2 Stylesheets and formatting 4](#_Toc152749535)

[3.3 Support for reversification and out of order verses 4](#_Toc152749536)

[4 Requirements 5](#_Toc152749537)

[5 Initial thoughts 5](#_Toc152749538)

[5.1 Support for both book mode and study mode 5](#_Toc152749539)

[5.2 Styling 6](#_Toc152749540)

[5.3 Reversification and out-of-order verses 6](#_Toc152749541)

[5.4 Compact representation 6](#_Toc152749542)

[5.5 Backwards compatibility 7](#_Toc152749543)

# Introduction

The present implementation of STEPBible does a lot of things very effectively. But at the same time there are a number of things it does not do. This document considers whether there might be an alternative implementation which can plug these gaps.

In particular …

1. For reasons discussed in section 3.1, we cannot presently generate indented paragraphs, something which translators very commonly require.
2. We have a fixed stylesheet which is applied to all texts. This may not always be appropriate; and it also means that where translators are able to specify what formatting they want (as with most DBL texts), we cannot respond to that.
3. We have no built-in support for the kinds of alterations associated with reversification, nor for out-of-order verses.

# Terminology

Before we begin, a little terminology which is used below.

STEPBible displays text in two different modes which, although superficially very similar, actually give rise to significantly different processing.

One I dub below **book-mode**. Book-mode is the on-screen equivalent of a printed book: the text is displayed in large chunks (typically entire chapters at a time), and the requirement is indeed to make it look as much as possible like its printed equivalent.

The other I will refer to as **study-mode**. In study mode, we typically display only a few verses (possibly as few as one) – perhaps as part of an interlinear display, as the target of a cross-reference, or by way of search results.

# Issues with the present implementation

## Paragraph-level formatting

Both USX and OSIS support milestone markers for verses – ie an empty <verse> tag at the start of the verse, marked with a sid (start-id) attribute which gives the verse reference, and another empty tag at the end of the verse marked with an eid (end-id) attribute.

They are defined in this way very specifically to permit semantic and formatting markup to run across verse boundaries, and all the texts I have investigated in any depth take advantage of this.

However, this complicates life significantly when operating in study-mode, because here we need to extricate verses from their context, and this is difficult if the verse comprises several different pieces of text lying under different parent nodes.

osis2mod appears to address this by converting some (all?) enclosing tags to self-closing form. This something which sets out as

<para style='p'>This is some text within a paragraph.</style>

ends up as

<para style=’p’/>This is some text within a paragraph.

This ensures that there is no conflict between markup and verse content,[[1]](#footnote-1) but at a cost, because it means there is no possibility of doing some of the things you would normally expect of a <div>. You can introduce vertical whitespace before the paragraph, and you can generate a first-line indent. But you can’t introduce vertical whitespace at the end of the paragraph, and you can’t apply a modified margin to the paragraph as a whole – and given that indented paragraphs are a very common requirement, this is a significant drawback.

## Stylesheets and formatting

In its present form, STEP works with a fixed stylesheet, which it applies to *all* texts. This has the advantage of being straightforward, and it does also mean that we’re able to impose a house style: if, for instance, text added by the translators appears italicised in one text, it will do so in all.

Against this, there is no guarantee that our stylesheet is always appropriate. We commonly italicise text, for instance. But quite possibly there are scripts which do not support italics, or where a sloping form of the text, even if it can be generated, is either meaningless or carries the wrong semantics. And also, at least where we are taking texts from DBL, the translators supply a styles.xml file indicating how the various USX tags are to be rendered.[[2]](#footnote-2) Perhaps we should pay at least some attention to this (particularly since the translators are more likely than are we to know what forms of rendering actually make sense to their target audience).

## Support for reversification and out of order verses

Different Bibles organise and label their content differently. One Bible may label the translation of a particular chunk of the Hebrew text as Gen 99:100, while another may label it as Gen 100:1. One may hold material in Daniel which another places in a deuterocanonical book. Sometimes a number of verses in one text are combined to form a single verse in another. Etc.

This complicates much of STEP’s added value processing. The verse vocabulary which we typically display against Gen 99:100 may need to be displayed against Gen 100:1 in another. Similarly, in an interlinear display, we cannot necessarily show Gen 99:100 in one text against Gen 99:100 in another – we might need to display it against Gen 100:1.

To address this, it is useful to adopt a common versification standard (we use NRSV), and then to use mapping tables to relate all texts to this common standard.

A further similar issue has come to light recently. The discussion in this section so far relates to reasonably common versification schemes – schemes which follow the Vulgate, the LXX, etc.[[3]](#footnote-3) However, latterly we have come across a number of Bibles from Biblica in which the verses have been deliberately placed out of order. They are still labelled appropriately, and still contain the text you would normally associate with the verse; but, say, v16 has been placed before v5.

# Requirements

1. Support for both book mode and study mode.
2. Support for per-text styling.
3. Support for reversification mappings and out-of-order verses.
4. Compact representation so as to minimise network bandwidth and storage requirements.
5. Encryption possible but optional.
6. Able to generate data which can be used by third parties.
7. Backwards compatible with existing modules.

The last of these requirements appears at the end of the list not because it is unimportant – it is, indeed, extremely important – but because I am not sure how it can be realised. I return to the matter in section 5.5.

# Initial thoughts

## Support for both book mode and study mode

At present I cannot think of a data structure which will support both of these modes directly and efficiently without pretty much duplicating the data, and duplicating the data would fly in the face of the requirement for a compact representation. I am therefore assuming we will have to adopt a structure in which one of the two modes is a poor relation.

One thing while I remember it …

The problem we are addressing here is largely the one of cross-boundary markup. Hitherto I’ve been trying to remove one possible source of cross-boundary markup by manipulating USX char-style tags. I have made the assumption (which I believe to be justified) that unlike para-level tags, a char-level tag has effect only upon the appearance of the text it contains: it won’t do things like introducing vertical whitespace before the tag, for instance. On that basis, if we have a char-level tag which spans a verse boundary, I believe it to be legitimate to split the tag. If, for instance, we have a char:it which spans the boundary, we can turn it into one char:it which applies to the text at the start of the verse and one which applies to the text at the end of the verse, leaving the verse boundary outside of either – and hence no cross-boundary markup:

**<char style='it'>**A very**<verse eid=…><verse sid=…>**important piece of text**</char>**

becomes

**<char style='it'>**A very**</char><verse eid=…><verse sid=…><char style='it'>**important piece of text**</char>**

Unfortunately, if we are to support per-text styling (and in particular if we are to take on board any formatting requirements specified by the translators, over which we will have no control), I am not sure we can continue to adopt this approach. USX, with its wide use of char and para tags, certainly *implies* that some elements (the char elements) will turn into HTML <span>, and that others (the para tags) will turn into <div>. However, we cannot now assume this. For instance, translators may want NT citations of OT passages to be split out into separate paragraphs, so what sets out as some flavour of char tag may end up having to be rendered as a div.

## Styling

We should still retain a default stylesheet, but should allow for overrides. USX tags should be converted in a highly consistent manner into HTML tags – eg char:it always turning into <span class='char\_it'>.

In working out what these mappings should be, we should be driven by what DBL does with its styles.xml file, since that’s the most common source of styling information at present.

## Reversification and out-of-order verses

I guess we’re going to need a two-way mapping table – text to NRSV and NRSV to text. We could at least assume that the absence of a mapping table means the text is already NRSV-compliant; and we could also make the assumption that if a given verse does not appear in the tables, it is because there is no need to apply a mapping to it.

As regards out-of-order verses, this is something of new territory. One thing I think it does rule out is the existing osis2mod approach which, as I understand it, does not label verses with their references, and simply assumes they are in order.

## Compact representation

I believe that the present Sword modules use ‘element-wise’ compression – for example the module may comprise of a number of separately zipped-chunks, perhaps one per chapter. The use of zipping promotes a compact representation; the use of element-wise zipping ensures that we don’t have to unzip the entire thing to access the content of just a few verses. I suspect this approach is probably as good as any.

## Encryption possible but optional

We can probably more or less carry through the existing implementation. The ‘optional’ part may not be of much value, though – see section 5.6.

## Able to generate data which can be used by third parties

**This won’t be feasible.** Or at least, it won’t be feasible unless we provide tools and / or guidance as to how to use the data, because we’ll be moving to our own proprietary representation. On this basis, the notion of *optional* encryption becomes rather pointless, because even in its unencrypted form, our data will not readily be usable by third parties.

## Backwards compatibility

I’m not at all sure this is really going to be feasible. At best, so far as I can see, we would effectively be trying to combine two entirely different pieces of software into one package. But even then, the modules currently being generated are just not set up to support the functionality we are looking for. It looks, therefore, as though we would effectively be moving to a completely separate new version of STEPBible.

# Implementation thoughts

Is book mode limited to displaying entire chapters, or do we need to cater for smaller book-mode chunks? And a related issue with study mode – what do we do about formatting (eg can we ignore it).

Data split into two parts – most / all tags in one part, along with non-canonical text; all canonical text held separately. Call these the structure table and the text table.

Separate slot in text table for each separate chunk of text. So if the canonical text of a verse starts under one parent tag and ends under another, you get two entries in the text table.

Could reduce the number of separate chunks if we say that any tags which fall entirely within the compass of a single verse could be stored as part of the canonical text of that verse. So, if a verse contained a couple of italicised words, for instance, the entire verse (prior text, italics tag, italicised text and following text) could just be stored as though it were the canonical text of the verse.

There are, apparently about 31,000 verses in the KJV. Probably many of these will not be split into more than one chunk. But allowing for the possibility that quite a few are, and also that we have to cater for DC, it might (frustratingly) be risky to assume a 16-bit index will be enough to access the text structure. However, it is not difficult to design an indexing mechanism such that most elements will be accommodated within a single 16-bit Unicode character, while still catering for the minority which may not.

When creating these structures, each canonical text item goes into a slot in the text table, and is replaced in the structure table by its index. Could enclose the index in some special tag – <T>, say – if that would help later processing.

In book mode, you’d run over the text to be displayed, replacing all of these indexes by their associated content.

If we assume book mode works only at the level of entire chapters, and that each chapter is zipped separately, then we don’t need additional indexing to obtain the chapter. Or rather we do, but it’s indexing which relates a chapter to the zipped chunk: it’s not indexing which has to work inside the structure and text tables.

This, I think, gives us all we need from the point of view of book-mode. (Except, perhaps, in respect of reversification – I think there are places where a request to display a given chapter would need to be translated into looking for a different chapter number in a different book.)

\* End of document \*

1. Actually, it doesn’t do so entirely. There are constructs which osis2mod can’t cope with in this way, and on occasion it appears to give up the unequal struggle. [↑](#footnote-ref-1)
2. Although whether this is used in any effective manner I am unsure. It may be that in many cases, this file is no more than standard boilerplate. [↑](#footnote-ref-2)
3. Even here, things are not entirely clear-cut. Very few Bibles follow any standard scheme absolutely. [↑](#footnote-ref-3)