**STEPBible module converter**

Module converter  
Config information



**Configuration Guide**

**Contents**

[1 Introduction 3](#_Toc201142709)

[2 Getting started 3](#_Toc201142710)

[3 Essentials 4](#_Toc201142711)

[3.1 Overall structure 4](#_Toc201142712)

[3.2 Comments and whitespace 5](#_Toc201142713)

[3.3 Simple assignment statements 5](#_Toc201142714)

[3.4 Parameter naming conventions 6](#_Toc201142715)

[3.5 Command-line parameters 7](#_Toc201142716)

[4 Not-so-essentials 8](#_Toc201142717)

[4.1 Include statements 8](#_Toc201142718)

[4.2 Defining parameters in terms of one another 9](#_Toc201142719)

[4.3 As-is lines 10](#_Toc201142720)

[4.4 The text converter environment variable 10](#_Toc201142721)

[4.5 Externally-supplied configuration data 10](#_Toc201142722)

[4.6 Special cases and caveats 11](#_Toc201142723)

[5 Built-in configuration 12](#_Toc201142724)

[5.1 Overview 12](#_Toc201142725)

[5.2 Detail 12](#_Toc201142726)

[6 Discussion 14](#_Toc201142727)

[6.1 What copyright definitely doesn’t affect 14](#_Toc201142728)

[6.2 What copyright may or may not affect 14](#_Toc201142729)

[6.3 What copyright definitely does affect 14](#_Toc201142730)

[7 The step.xlsx template 15](#_Toc201142731)

[8 Maintenance 16](#_Toc201142732)

# Introduction

|  |
| --- |
| * There’s a lot you *can* configure in the converter. * That doesn’t mean there’s always a lot you *have* to configure. * Some things are unavoidable. The converter has to know how you want a text to be processed, and needs information about its provenance etc. * But there are an awful lot of defaults built into the system which you will need to override only rarely if at all. * And where you have many texts with characteristics in common, you can share configuration data. * In what follows, things you *have* to know are in pink boxes. * Things you might possibly get away without knowing are in green. However, even if you don’t rely upon these, it might be useful at least to skim them, so you know roughly how things work. |

# Getting started

|  |
| --- |
| * Every text has to have its own folder, which I refer to here as the ***root folder***. * I recommend putting all of the root folders you work under a single folder (directly or indirectly). This will make it easier to control things and move data should you need to. However, this is not essential. * The root folders have to have a names like Text\_eng\_NIV\_xxx, where eng is the 3-character language code, NIV is the English abbreviated name of the text, and xxx is one of [public, step, or publicStep], plus optionally onlineOnly, in any order. (xxx is not case-sensitive.) * The xxx part tells the processing whether we can generate a STEPBible-format module from the text, a public-facing text, or both; and it also highlights those few texts which we are not permitted to make available offline * Under the root folder you create a folder called Metadata. * Historically, you have stored a text file called step.conf in that, to act as the base of the configuration information – and you can still work this way if you want. However, recent changes favour the use of a spreadsheet called step.xlsx in place of step.conf, and there is a template you can use for this purpose. This document is about what goes into those files. * If you are working with data from DBL and want to be able to pick up DBL metadata automatically, the Metadata folder is also where you store the DBL metadata.xml and license.xml files. * (There are other things you need to know about setting up the root folder before you can use the converter; here we are concerned only with those things relevant to supplying configuration information.) |

# Essentials

## Overall structure

|  |
| --- |
| Configuration data is taken from the following places:   1. From the command line used to invoke the converter. 2. From the configuration files you create. 3. From externally-supplied metadata files where possible. (At present we cater only for DBL metadata files.) 4. From various files built into the Resources section of the converter JAR file. 5. From the environment variable **stepTextConverterParameters** if it is defined.   Because there are so many places in which a variable may be defined, it is always possible that there may turn out to be more than one definition for it.  This means that precedence matters, and the list above is (more or less) in precedence order – command line settings take priority over everything else, and so on. I say ‘more or less’ because you do have some measure of control over this in the bits which you define yourself as we’ll see shortly.  \*  All texts have to have a file called **step.xlsx** (or step.conf if you choose to continue working that way) in their Metadata folder[[1]](#footnote-1). The processing always starts with this file when working out what to do. The template described later helps you create step.xlsx.  Putting to one side the settings built into the converter JAR, you *can* place all of the configuration information into step.xlsx / step.conf if you wish. Very often you won’t want to do this, however – you will wish to put different parts of the configuration data into different files, for example so that you can share common information among a number of texts.  The ‘include’ mechanism discussed in section 4.1 lets you do this – it provides a way whereby one file can point out to another.  Having said this, no particular arrangement is forced upon you — there is no *requirement* to split data in this way, and no particular significance is ascribed to any arrangement you adopt.  \*  The language used to describe configuration data is made up of:   * Assignment statements, which associate values with parameter names. * ‘include’ statements, which let one file include the contents of another. * Comments.   If you create your own text files (\*.conf), these are the kinds of statements you put in them. If you use step.xlsx as the root of your configuration data, its content is used to generate statements like these. |

## Comments and whitespace

|  |
| --- |
| * Comments start with ‘#!’. (I’d prefer to use just ‘#’, but an unfortunate decision a long time ago makes that impossible.) * They may appear on lines to themselves or at the end of other configuration directives. * Everything from the start of the comment marker to the end of the line is ignored. * After removing comments, leading and trailing whitespace on a line is ignored. * And after doing all of this, completely blank lines are ignored. |

## Simple assignment statements

|  |
| --- |
| Assignment statements associate a given value with a parameter name.  In simple assignment statements, the value you assign to the parameter is a piece of fixed text. Simple assignment statements come in two forms:  **stepSong=Jingle bells** (vanilla assignment)  **stepSong#=Jingle bells** (forcible assignment)  Both of these assign the value ‘Jingle Bells’ to the parameter *stepSong*. The difference (signalled by the presence of the # in the second example) lies in the way they deal with multiple assignments to the same parameter.  \*  After includes have been expanded out, we essentially have one large flat text file, which is processed in order from start to finish. It is perfectly possible that in this expanded data there may be multiple statements each of which assigns a value to the same parameter.[[2]](#footnote-2)  If the expanded text contains a number of *vanilla* assignments to the same parameter (and no forcible assignments), it is the *last* one which wins – the others are ignored.  If the expanded text contains *any* forcible assignments, the *first* one to be encountered wins – it supersedes any vanilla assignments which may already have been encountered, and it prevents any further assignments, vanilla or forcible.  \*  Most commonly, you will want to be sure that any assignments you make yourself take priority over everything else.  The way to achieve this is to use forcible assignments, and to position them *before* any include statements, since it is always possible that included text might interfere with what you are trying to achieve. If you use step.xlsx, this is done for you.  Nonetheless, there may occasionally be a case for using vanilla assignments. You might, for instance, want to set up a .conf file containing defaults of your own to be shared amongst a number of texts. In this case, you’d set up the defaults as vanilla assignments, and then use forcible assignments for the overrides.  \*  Returning to the assignment statements above, you can also assign an empty value to a parameter (I’ll just show a forcible assignment here, but both work the same way):  **stepSong#={empty}**  Note that spaces immediately after the equals sign and at the end of the right hand side are ignored. If you need a space in either of those places, specify it as *{space}*:  **stepSong#={space}{space}Happy Birthday{space}{space}**  And finally, you can continue long strings across multiple lines if you wish, by ending all lines but the last with a backslash. Whitespace prior to a backslash is ignored, but whitespace at the start of a continuation line is respected.  \*  The examples above all assign fixed values to parameters. Sometimes it may be useful to construct one parameter out of a combination of other ones. See section 4.2. |

## Parameter naming conventions

|  |
| --- |
| * Despite my best endeavours, naming is a bit of a mess. * Most parameters have names starting with ‘step’. * A few, which turn directly into Sword configuration parameters and are not subject to internal processing, have names starting ‘sword’. * Ones which are assigned fixed values internally and are never subsequently changed have names starting ‘const’. * Ones whose values are normally assigned as a result of internal calculations have names starting ‘calc’. However, this convention may be of only limited value, because, for example, a parameter which can be calculated from the supplied metadata on DBL texts may have to be supplied by the user on non-DBL texts, and I probably haven’t named these very consistently.   \*   * In general there is very little point in making up your own parameter names, because the processing is set up to make use only of parameters which it already knows about. Occasionally, though, you may be assigning a complex value to something, and may find it useful to store intermediate values. Give these intermediate values name starting with underscores – such names are guaranteed not to clash with anything else. |

## Command-line parameters

|  |
| --- |
| A few configuration parameters can be – or must be – supplied on the command line: things like *rootFolder*, which tells the converter which text it is to process, *releaseNumber*and *history*. (The full list is documented elsewhere, and can also be obtained by giving the parameter -help on the converter command line.)  It is just possible you may want to refer to the values of these when assigning values to parameters (we discuss how to base one parameter upon another in section 4.2). When you do that, you need to add ‘step’ on to the name and change the capitalisation. For example, there is a command-line parameter *targetAudience*, which says whether a given run is to produce a STEPBible-only module or a public one. If you need to refer to this, you use the name *stepTargetAudience*. |

# Not-so-essentials

## Include statements

|  |
| --- |
| There are two flavours of include statement –  **$include <path>**  **$includeIfExists <path>**  They differ only in that the former aborts processing if the file it is looking for does not exist.  \*  I recommend using .conf as the extension for configuration text files, although nothing relies upon this. Paths should be in Linux format – ie separators should be ‘/’, and you can, to a limited extent, use ‘.’ and ‘..’ You should assume that paths are case-sensitive.  Note that ‘include’ statements can *only* be used to include text files – you can’t ‘include’ spreadsheet files. (The only spreadsheet file supported is step.xlsx, and the converter looks for that one automatically, so there is no need to include it from anywhere.)  \*  The file path can be specified in a number of different ways:  **Starting with the string ‘@find’ (eg $find/theFile.conf):**  You should always use this option unless you have very strong grounds for not doing so. This looks for ‘theFile.conf’ in each of the following folders, in this order: the text’s root folder; the Metadata folder; and any folders defined in the environment variable *stepTextConverterParameters* against *stepConfigFolderPaths* (see section 4.4).  **Starting with the string ‘@jarResources’ (eg $jarResources/theFile.conf).**  This is really reserved for internal use. It says that the file resides in the Resources section of the converter JAR file. In the normal course of events you won’t need to use this option, because these files are picked up automatically.  **Starting with any absolute path (eg somethingOrOther/theFile.conf).**  This lets you specify an absolute path. Don’t leave this lying around in your data, because it makes it non-portable. But you may occasionally find it useful for debugging purposes.  In all cases, you can add further levels below the bit in red above, so you can have eg @find/FolderA/FolderB/theFile.conf.  You can also assemble paths out of other configuration parameters – eg @aParm/theFile\_$anotherParm.conf. See section 4.2.  \*  Include statements can be nested to any reasonable depth. Processing proceeds as though each statement were replaced by the content of the file to which it points. The resulting flat text is then processed in order. |

## Defining parameters in terms of one another

|  |
| --- |
| The assignment statements discussed above assign a fixed value to a parameter. Sometimes it is convenient to build one parameter out of others.  The simplest way to do this is to include something like $myParm on the right hand side of a definition. If you have:  **stepName#=Ethelred**  **stepHi#=Hi $stepName**  the effect is to assign to *stepHi* the value ‘Hi Ethelred’.  \*  You can also use *$choose* to select the first non-blank value from a list:  **stepHi#=Hi, $choose($stepFriendA, $stepFriendB, …)**  will assign to *stepHi* a welcome message to the first non-blank friend.  And you can optionally supply a default value, which will be returned if the other arguments are all blank / undefined :  **$choose($stepFriendA, $stepFriendB, "Sob! No friends")**  \*  $ references may be nested to any depth – ie you can have a statement containing a $ reference, which expands to another statement containing a $ reference, and so on.  The values of parameters are not evaluated until the processing actually requires them. In the first example above, stepHi will be assigned ‘Hi Ethelred’ if stepName has the value Ethelred at the point stepHi is requested, even if stepName has already been assigned various different values up to that point.  Moreover, once a value has been requested, that value is permanently assigned to that parameter. After accessing stepHi, it would continue to be ‘Hi Ethelred’ even if stepName were changed again subsequently.  Note, incidentally, that *$choose* uses short-circuit evaluation: in the example above, if *$stepFriendA* returns a value, *$choose* will not evaluate *$stepFriendB*. |

## As-is lines

|  |
| --- |
| All of the directives discussed to date feed into the converter’s processing, and it is the converter which decides how they are used.  Sometimes, though, you may need to pass some information directly through to the Sword configuration file unchanged. (A particular example of this arises from the need to specify what previous modules a given revised module may render obsolete.)  For this you use:  **CopyAsIs=Some Sword config directive or other**  You can have as many of these lines as necessary. |

## The text converter environment variable

|  |
| --- |
| You can, if you wish, set up an environment variable – stepTextConverterParameters – to hold configuration data, although there is no *requirement* to do so.  The content of the variable is simply a standard set of assignment statements, as discussed above, separated by semicolons. If you wish a value to *contain* a semicolon, give it as \;  You can define anything you want here, but the following are likely to be the most relevant:   * **stepTextConverterOverallDataRoot**: The -rootFolder parameter you give on the command line to say which text to process can give either an absolute path or a relative one. Relative paths are taken as being relative to the value of **stepTextConverterOverallDataRoot**. Using this setting may make it easier to move data en masse – if you move your texts, you need only change the setting to point to the new location, and the existing command lines need not change. * **stepOsis2ModFilePath**: The path to the osis2mod executable. You have to supply this *somewhere*; putting it into the environment variable is probably as good a place as any. * **stepConfigFolderPaths**: A comma-separated list of paths to folders which may hold shared configuration information. When trying to resolve @find on include statements, these folders are examined in order, looking for the target file. |

## Externally-supplied configuration data

|  |
| --- |
| There are some text repositories which we deal with very commonly – notably DBL at the time of writing.  For DBL (and currently *only* for DBL) we have processing which – at least in theory – can take configuration and licensing information for the metadata and licence files which they supply. This means you may not have to fill in all of the data which would be required for non-DBL texts.  For these, you simply indicate that this is indeed a DBL text, which you do by setting stepExternalDataFormat to DBL; and you ensure that metadata.xml and license.xml are placed in the Metadata folder.  A few caveats though:   * In quite a number of cases we have chosen to override the values available in the DBL metadata. * The supplied metadata tends not to be used in a very consistent manner. It is probably a good idea to check the results of any run which depends upon supplied metadata.   You may occasionally want to check to see which metadata fields are being accessed in order to furnish the various configuration parameters. The file PerTextRepositoryOrganisation / Dbl.conf in the Resources section of the JAR file gives details. This relies upon another $-function ($getExternalXml) to extract data from the supplied XML files. |

## Special cases and caveats

|  |
| --- |
| When you reference another parameter using $myVar as described above, the name of the parameter is taken as all of the ‘word’ characters (letters, numbers, digits and underscores) from the $ up to the first non-word character.  Sometimes this does not quite work. Suppose you have a variable *abbrev* which can have the values Sun, Mon or Tues, and you want to use this to generate Sunday, Monday or Tuesday. The obvious approach would seem to be to use $abbrevday, but that won’t work, because abbrevday will be taken as a parameter name. You need some way of marking the end of the actual name. To achieve this, end the variable name with a double dollar sign – $abbrev$$day. (This is needed only in cases like this, although there is no harm in using it even where it is not needed.)  There are also a couple of caveats to be taken into account:   * Any printer’s curly double quotes you may include in the configuration data are converted to straight quotes (ie there is no point in using printer’s double quotes and hoping they will end up in the output: they won’t). * You can’t have standalone $-signs within any text values (or you *probably* can’t – I haven’t checked this). $-signs are always taken as signifying that we have a parameter which needs to have some kind of processing applied to it. |

# Built-in configuration

## Overview

|  |
| --- |
| The converter is very highly configurable. Much of the configuration data is given over to default settings which you will hardly, if ever, need to change, and this information resides in files stored in the Resources section of the converter JAR file.  Just occasionally, you may need to override one or two deefinitions, however, so it may be useful to know what is where. The various files are listed below. Most contain extensive head-of-file documentation.  Files with names ending .conf are in our proprietary configuration format. Files ending in .tsv are tab-separated variable format. Files ending .txt are plain text. |

## Detail

|  |
| --- |
| **PerTextRepositoryOrganisation / Dbl.conf**  Information which describes how to extract configuration data from the DBL metadata and licence files. |

|  |
| --- |
| **ReferenceFormats / referenceFormatVernacularDefaults.conf**  Describes how vernacular references are formatted. By default we assume they simply follow the UBS English scheme. It may occasionally be useful to override this if we wish to output our own references. In practice, though, it will be painful enough to obtain the necessary information that we are unlikely to do it very often. |

|  |
| --- |
| **basicDefinitions.conf**  Simple common definitions – things like standard rubric for CC licensing. Saves having to come up with the same material repeatedly. |

|  |
| --- |
| **bookNames.tsv**  Book names and abbreviations as used in various different naming schemes. Tab-separated variable format. |

|  |
| --- |
| **commonRoot.conf**  Every configuration scheme sooner or later must include this file. It arranges things so that if vital configuration information is missing, errors will be generated; and it arranges to include the appropriate files from the Resources section, so as to ensure we have all the necessary defaults. |

|  |
| --- |
| **configDataDescriptors.tsv**  Describes all the various different kinds of configuration data we may use, partly for documention purpose, and partly for checking and debugging. This data comes from a separately-maintained spreadsheet – configDescriptions.xlsm. |

|  |
| --- |
| **countryNamesToShortenedForm.tsv**  There are a few countries which have long official names, but for which a shorter form is commonly used. This file maps long to short. |

|  |
| --- |
| **externalDataPaths.conf**  URLs of various external sites upon which the converter relies or of which it may need to make mention. |

|  |
| --- |
| **isoLanguageCodes.tsv**  Maps between 2-character codes, 3-character codes, language names, etc. |

|  |
| --- |
| **osis2modVersification.txt**  Details of the various Crosswire osis2mod versification schemes. |

|  |
| --- |
| **standardBookNames.conf**  UBS long, short and abbreviated book names. |

|  |
| --- |
| **strongsCorrections.txt**  Maps from commonly used incorrect Strong’s numbers to the corrected form. |

|  |
| --- |
| **swordTemplateFile.conf**  A template used to create Sword configuration files. |

|  |
| --- |
| **usxToOsisTagConversionsEtc.conf**  Pretty much what it says on the tin. |

# Discussion

One of the more complicated things to wrap your mind around at present is the matter of how the copyright status of a text interacts with various aspects of its processing.

Until recently, this was driven entirely by the stepIsCopyrightText parameter. However, it turned out that this could no longer cope with all of the things we needed to do, so the processing has been revised as discussed below.

The things we need to control are:

* Are we permitted to make a public version of the module?
* Do we need to apply versification processing?
* Can we add versification-related footnotes?
* Do we admit to the possibility that the text may have been changed slightly during processing?

## What copyright definitely doesn’t affect

* We *always* make a STEPBible version of each module.
* STEPBible versions are *always* subject to versification processing.

## What copyright may or may not affect

* We may or may not make a public module.
* We always do so for non-copyright texts.
* And we may sometimes make a public module even for copyright texts. This may seem odd, but at least one text owner has given us permission to make a public module from a copyright text, and we are hoping that others will too. In view of this, the decision as to whether to create one or not cannot be driven directly by the copyright status of a text.

## What copyright definitely does affect

* We work on the assumption that the licence conditions for copyright texts normally preclude the addition of footnotes, so we never add any.
* On non-copyright texts, we add to the copyright page a brief note explaining that we may have had to make minor changes to the text in order to handle it.
* We don’t do this on copyright texts. The changes we make are hopefully indeed *very* minor, and it would seem unfortunate to draw attention to possible minor changes on copyright texts.

# The step.xlsx template

You can obtain a copy of the step.xlsx template from the Resources section of the converter JAR file. (JAR files can be opened using standard zip software.)

Simply fill in the appropriate details and save it in the Metadata folder under the root folder for the text.

Fields are colour-coded as an aid to identifying those which must be filled in, those which often might be filled in, and those which can probably be left to be defaulted by the processing.

The colour-coding should not be taken as definitive, because it is difficult to cater for all possibilities. For example, a field may be mandatory on texts of a certain kind, but optional or not required at all on different kinds of texts (eg those originating from DBL) – and it is difficult to show these kinds of nuances when all you can do is colour the cell.

Some of the cells can be filled in from DBL, and the spreadsheet indicates where this is the case. If you opt to take data from the DBL metadata files, do check the result – different translators interpret the rules for metadata files differently, so processing which picks up the correct information for one text may fail to do so on another.

# Maintenance

The configuration data is administered using a file configDescriptions.xlsm (and yes, sorry about that, but this one is definitely an *Excel* spreadsheet).

The sheet *AllConfigurationParameters* in this file contains a full list of all of the configuration parameters used by the converter. You need to change this to reflect any changes to the parameters which the converter uses. If you make any changes, you need to copy the content of the named range *ForUseInConverter*, and use it to replace the contents of configDataDescriptors.tsv in the Resources section of the converter JAR file.

Also if you make changes to *AllConfigurationParameters* you need to generate a new step.xlsx. There is a button on *AllConfigurationParameters* which generates a suitable worksheet for the purpose, and stores it in configDescriptions.xlsm as *ForStepConfig*.

You need to copy *ForStepConfig* to a new file, save it as step.xlsx, and store it in the Resources section of the JAR file. (Note that this is an .xlsx file, not .xlsm, so hopefully it should work in tools other than Excel – for example Google Sheets or LibreOffice Calc.

**\* End of document \***

1. If you have both, step.xlsx is used in preference, and step.conf is ignored. However, I strongly recommend *not* having both – it will be confusing. [↑](#footnote-ref-1)
2. This is how defaulting is handled, in fact. Files in the Resources section of the converter JAR file contain default settings which take effect unless you override them, as described here. [↑](#footnote-ref-2)