**CharacterCounter: a python module for checking character counts in GLS conversion projects**

**Version 1.3**

James McCracken, 7 September 2013

jmccracken@fastmail.fm

**Contents**

[1 Introduction 1](#_Toc366400945)

[2 Limitations 1](#_Toc366400946)

[3 Usage 2](#_Toc366400947)

[4 Configuration 2](#_Toc366400948)

[4.1 General configuration 2](#_Toc366400949)

[4.2 Configuring the source data 2](#_Toc366400950)

[4.3 Configuring the converted (output) data 3](#_Toc366400951)

[4.4 Setting parameters 3](#_Toc366400952)

[5 Suppressed elements 3](#_Toc366400953)

[6 Constant adjustments 4](#_Toc366400954)

[7 Attribute adjustments 4](#_Toc366400955)

[8 Entry IDs 5](#_Toc366400956)

[8.1 Autodetection 5](#_Toc366400957)

[9 Entry alignment 5](#_Toc366400958)

[10 Methods 6](#_Toc366400959)

[10.1 total\_variance() 6](#_Toc366400960)

[10.2 log\_variances() 6](#_Toc366400961)

[11 Technical requirements 7](#_Toc366400962)

# Introduction

The character counter is a tool to check that the text content of each converted entry matches (approximately) the text content of the corresponding entry in the source data.

Unexpected variances are highlighted as possible indicators of an error in the conversion process.

# Limitations

* The character counter only works when the source data is XML. Source data in other formats will need bespoke character-counting methods.
* All whitespace (including newlines and tabs) is stripped from both source and converted before comparing entries. Hence variance in whitespace (spaces lost or added) will not be registered.

The parameters passed to the character counter object (see section 4) compensate for various ways in which the source text may be expected to differ from the converted text. However, the code may sometimes need to be adapted to compensate for other factors not anticipated here.

# Usage

To run interactively in a Python shell:

>>> from charactercounter import CharacterCounter

>>> cc = CharacterCounter(source\_file='/path/to/source.xml',

converted\_file='/path/to/converted.xml',

source\_entry\_tags=['ENTRY',],

suppressed\_elements=('EtymHid', 'CatHid'),

constant\_adjustments=[('DATE', 1),],)

>>> cc.total\_variance()

Source text: 2002986 characters

Converted text: 2000150 characters

Variance: -2836 characters (-0.14%)

>>> cc.log\_variances(percentage\_threshold=5)

The module can also be run directly (python charactercounter.py), in which case the script at the end (in the ‘if \_\_name\_\_ == '\_\_main\_\_'’ block) will be executed. (The code is provided with some sample data to demonstrate this.)

# Configuration

Configuration parameters are passed as keyword arguments.

## General configuration

* source\_file: the path to the file containing the source XML;
* converted\_file: the path to the file containing the converted XML;
* realign: set to True or False. If True, entries are automatically realigned based on ID numbers; see section 9 below. Defaults to False;

## Configuring the source data

The following keyword arguments specify details of the source data:

* source\_entry\_tags: a list of tags used to identify an entry in the source data. Defaults to [‘e’,].
* source\_constant\_adjustments: a list of adjustments made for extra characters deriving from particular elements. Defaults to [] (empty list).
* source\_attribute\_adjustments: a list of adjustments made for extra characters derived by turning attributes into character data. Defaults to [] (empty list).
* source\_suppressed\_elements: a list of elements to be ignored. Defaults to [] (empty list).
* source\_headword\_tags: a list of tags used for the headword in the converted data. Defaults to [‘hw’,].
* source\_id\_attribute: the name of the attribute holding the entry ID value in the converted data.

## Configuring the converted (output) data

The following keyword arguments specify details of the converted data. These correspond directly to the source-data parameters listed in the previous section (and have the same defaults):

* converted\_entry\_tags
* converted\_constant\_adjustments
* converted\_attribute\_adjustments
* converted\_suppressed\_elements
* converted\_headword\_tags
* converted\_id\_attribute

## Setting parameters

Configuration parameters may be passed as keyword arguments when creating the CharacterCounter object, or may be set as individual attributes after creating the CharacterCounter object. Or a mixture of the two, e.g.:

>>> from charactercounter import CharacterCounter

>>> cc = CharacterCounter(source\_file='foo.xml',

converted\_file='bar.xml')

>>> cc.source\_entry\_tags=['ENTRY',]

>>> cc.suppressed\_elements=('EtymHid', 'CatHid')

source\_file and converted\_file are required; the others are optional. (Defaults will be used if any parameters are not set explicitly.)

# Suppressed elements

The suppressed\_elements parameter specifies a list of elements that are to be ignored. These elements, and all child elements, will be stripped from the entry node before any counting is done.

* source\_suppressed\_elements specifies elements to be ignored in entries in the source data.
* converted\_suppressed\_elements specifies elements to be ignored in entries in the converted data.

Values may be simple tag names, e.g. ‘EtymHid’, or full XPath expressions, e.g. “\*[@media=’online’]”, or a mixture of the two.

For example, the following parameter:

source\_suppressed\_elements=('EtymHid', “.//\*[@media=’online’]”)

means that all <EtymHid> elements (and any child elements) will be stripped from the source entry node. All elements matching the XPath expression “\*[@media=’online’]” will also be stripped. (Note that XPath expressions are evaluated relative to the parent entry node.)

This is useful if entries contains metadata or other non-displaying elements that should not contribute to the character count.

Defaults to an empty list.

# Constant adjustments

The constant\_adjustments parameter is a list of tuples, each tuple consisting of a tag name and an integer value (positive or negative). The integer value will be added to the source entry’s character count every time the element is encountered.

* source\_constant\_adjustments specifies values for entries in the source data.
* converted\_constant\_adjustments specifies values for entries in the converted data.

This is useful if the conversion process generates new characters for each occurrence of a given element in the source data.

For example, if the conversion process generates opening and closing square-brackets (2 extra characters) for each occurrence of <etymology> in the source entry, and generates a bullet point (1 extra character) for each occurrence of <bullet> in the source entry, then the following argument:

source\_constant\_adjustments=[('etymology', 2), ('bullet', 1)]

would ensure that corresponding values are added to the source entry’s character count for every occurrence of <etymology> and <bullet>, meaning that the source count and converted count are kept in line with each other.

Note that the character counter strips all whitespace from both source and converted entries. So there’s no need to add constant adjustments for generated spaces. E.g. if each occurrence of <bullet> generates a bullet point plus a space, then this should be accounted for as one extra character (i.e. source\_constant\_adjustments=[('bullet', 1)]) - since the generated bullet point is countable, but the space is not.

Defaults to an empty list.

# Attribute adjustments

The attribute\_adjustments parameter is a list of tuples, each tuple consisting of a tag name and an attribute name. When an element is encountered with the given tag and containing the given attribute, the length of the attribute value is added to the entry’s character count.

* source\_attribute\_adjustments specifies values for entries in the source data.
* converted\_attribute\_adjustments specifies values for entries in the converted data.

This is useful if the conversion process turns an attribute in the source into character data in the output.

For example, if the conversion process turns <wordclass value='adjective'/> into character data in the output, then the following argument:

source\_attribute\_adjustments=[('wordclass', 'value')]

would ensure that that the length of the wordclass/@value attribute was added to the source entry’s character count, meaning that the source count and converted count are kept in line with each other.

Defaults to an empty list.

# Entry IDs

Entry IDs are used to identify particular entries in the variance report (see section 10.2), and to align entries (see section 9).

Entry IDs are assumed to be stored as attributes on the entry tag. The attribute name can be specified through the source\_id\_attribute and converted\_id\_attribute parameters:

* source\_id\_attribute: specifies the name of the attribute used in the source data, e.g. source\_id\_attribute=’id’
* converted\_id\_attribute: specifies the name of the attribute used in the converted data, e.g. converted\_id\_attribute=’xrid’

## Autodetection

If source\_id\_attribute or converted\_id\_attribute are left undefined, the script attempts to autodetect the ID value by checking the entry tag for the following attributes (in this order):

* xrid
* id
* ID
* lexid

The first non-null value found is taken to be the entry’s ID.

# Entry alignment

By default, source data and converted data are assumed to be aligned, i.e. each entry in source data has a corresponding entry in the converted data; and entries are in the same order. Hence, when comparing entries, the first entry in the source data will be compared to the first entry in the converted data, the second entry to the second entry, and so on.

If this is not the case – for example, if some entries in the source data are not included in the converted data, or if entries have been re-ordered in any way, then the character counter will need to be instructed to realign entries. Do this by passing realign=True as a keyword argument when initializing the CharacterCounter object:

>>> from charactercounter import CharacterCounter

>>> cc = CharacterCounter(

source\_file='foo.xml',

converted\_file='bar.xml',

source\_entry\_tags=['ENTRY',],

realign=True)

Automatic realignment works by pairing up entries with matching ID numbers. For each entry in the source data, the process checks the converted data for an entry with the same ID. If no matching entry is found, the source data entry is paired with a dummy (null) entry. This will be noted in the variance log file.

Realignment depends on the source\_id\_attribute and converted\_id\_attribute parameters having been set correctly. The values of these attributes must be identical in the source and converted data.

# Methods

The two main methods are total\_variance() and log\_variances():

## total\_variance()

total\_variance() (no arguments) returns a three-line string giving a short summary of overall variance between the two documents:

>>> cc.total\_variance()

Source text: 8715305 characters

Converted text: 8715713 characters

Net variance: 408 characters (0.0047%)

Characters have been lost in 16593 entries (24% of all entries)

(Total characters lost: 112847)

Characters have been gained in 9094 entries (13% of all entries)

(Total characters gained: 113255)

Net variance is positive if the converted file contains more characters than the source, negative if it contains fewer.

## log\_variances()

log\_variances() creates a CSV file recording every entry where variance was observed (any non-zero variance, or variance above a certain threshold), with details of the magnitude of the variance and some guide to how the entries differ from each other.

The following options may be passed to log\_variances() as keyword options:

* csv\_file: path of the output file. Should have a ‘.csv’ extension. Defaults to a file called ‘variances.csv’ in the present working directory
* absolute\_threshold: integer value specifying the minimum absolute variance; entries with variance below this threshold will not be logged.
* percentage\_threshold: integer or float value specifying the minimum percentage variance; entries with variance below this threshold will not be logged.

Example:

>>> cc.total\_variance(csv\_file='/path/to/cc\_output.csv',

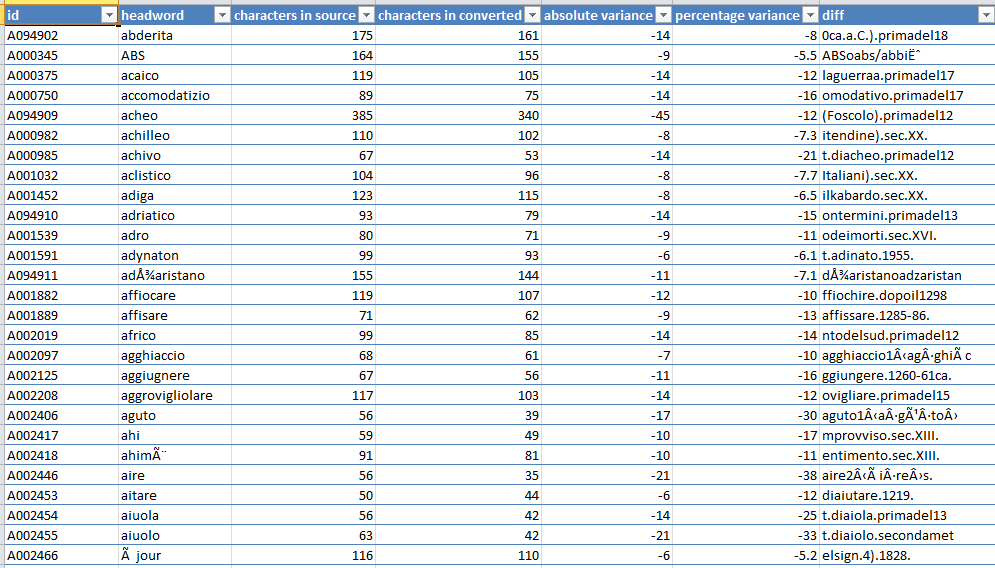
absolute\_threshold=5, percentage\_threshold=2)

absolute\_threshold and percentage\_threshold are useful in cases where some legitimate variance is to be expected, so you’re only interested in entries with significant variance.

Note that if both absolute\_threshold and percentage\_threshold are set, then any entry that exceeds *either* threshold will be logged.

The output CSV file has the following columns:

* A: entry ID (@xrid value), as given in the converted version;
* B: entry headword, as given in the converted version;
* C: number of characters calculated for the source version;
* D: number of characters in the converted version;
* E: absolute variance (number of characters) - positive if the converted entry contains more characters than the source, negative if it contains fewer;
* F: percentage variance;
* G: a short snippet of text showing the first point in the entry where a difference between the two entries was observed.



The text snippet in column G many not always be accurate (because of whitespace stripping and other manipulations). But it’s often useful to give a rough idea of where the first difference occurs.

# Technical requirements

The character counter is written in Python 2.7, but should be backwards-compatible at least to Python 2.6, and forwards-compatible to Python 3.x.

It requires the lxml library to be installed (<http://lxml.de/>). (I’m using lxml version 2.3.6, but any 2.x or 3.x version should be okay – it’s not doing anything fancy.)