Developer Manual

for

Lingua::Translit

Transliterate text between various writing systems.

This developer manual covers the 0.x series of Lingua::Translit.

Lingua Systems
NATURAL
LANGUAGE
PROCESSING

Lingua::Translit Developer Manual, published December 30, 2009.

Copyright © 2009, Lingua-Systems Software GmbH

Lingua-Systems Software GmbH, Wiesenstraße 34, 44653 Herne , Germany,
info@lingua-systems.com

All rights reserved, especially changing or publishing parts of this manual needs prior written permission of the copyright owner.

The rights to reproduce and publish unchanged copies in any form, to translate or to present the manual are granted.

Mentioned hard- and software as well as companies may be trademarks of their respective owners. Use of a term in this manual should not be regarded as affecting the validity of any trademark or service mark. A missing annotation of the trademark may not lead to the assumption, that no trademark is claimed and may thus be used freely.

Great effort has been made in writing this manual. However, faults can not be excluded in general. For any loss or damages caused or alleged to be caused directly or indirectly by errors or omissions in this manual, the authors and the publisher assume no responsibility and can not be held liable. Neither can the authors or the publisher be held liable for the content or changes of content concerning the linked websites. The links have been carefully chosen and proved at the preparation of the manual.

If you have problems using the links or get aware of any faults, feel free to give a brief hint on it via *perl@lingua-systems.com*.

Contents

1 Conventions used in this manual

Every non absolute path is relative to *Lingua::Translit*'s source code directory.

2 Adding transliteration tables

If you want to add a new transliteration to *Lingua::Translit* just…

→ …write an XML file (the "transliteration table")

→ …build a development version containing your table

→ …write and run some tests to check if your transliteration is working as expected

→ …integrate your table into the set of upstream tables and consider contributing it

3 Writing a transliteration table

Each XML transliteration table consists of meta data and a set of transliteration rules.

The *meta data* tags cover the name of the transliteration, a short description and the information whether the transliteration can be used in both directions. For example:

```
<name>DIN 1460</name>
<desc>DIN 1460: Cyrillic to Latin</desc>
<reverse>true</reverse>
```

The *rules* can be simple one to one mappings:

```
<rule>
    <from>X</from>
    <to>Y</to>
</rule>
```

…but you can also specify a *context* in which the rule should be evaluated only:

```
<rule>
    <from>A</from>
    <to>B</to>
    <context>
        <after>x</after>
        <before>y</before>
    </context>
</rule>
```

To get an easy start, you can copy the file *xml/template.xml* (see appendix A), rename it as needed and edit it right away. Additionally, *xml/Common_DEU.xml* may be used as a complete example.

Although editing an XML file is technically quite easy, some things have to be considered. The most important thing to keep in mind is that the rules are applied *in sequence* - one after another. Therefore the order of rules is important if you specify a context or transliterate multiple characters.

3.1 Unicode notation

If you are determining characters that are non-ASCII characters, use an entity that represents the Unicode code point in hex-notation to specify them and leave a comment on the character.

```
<rule>
    <from>&#x0410;</from>  <!-- CYRILLIC CAPITAL LETTER A -->
    <to>A</to>
</rule>
```

This assures that the correct character is transformed and it can be exactly determined, if it is not represented correctly.

3.2 Specifying a context

The context is evaluated as a Perl regular expression. So for specifying the context *literal ASCII characters*, *entities* or *meta characters* can be used.

If a character has two mappings depending on the context, the context-sensitive rule must be applied before the context-free rule. Otherwise every character is replaced at once through the context-free rule and the context-sensitive rule will never match.

1. rule:

```
<rule>
    <from>&#x0393;&#x03BA;</from> <!-- GREEK CAPITAL LETTER
                                       GAMMA & SMALL LETTER
                                       KAPPA -->
    <to>Gk</to>
    <context>
        <after>\b</after>         <!-- word initial -->
    </context>
</rule>
```

2. rule:

```
<rule>
    <from>&#x0393;&#x03BA;</from> <!-- GREEK CAPITAL LETTER
                                       GAMMA & SMALL LETTER
                                       KAPPA -->
    <to>Nk</to>
</rule>
```

The following pattern matching contexts are available:

→ `<after>`

if the transliteration rule should only be applied after a certain character (corresponds to Perl's *lookbehind*)

→ `<before>`

if the rule should only be applied before a certain character (corresponds to Perl's *lookahead*)

→ `<after>` & `<before>`

if the rule should only be applied if the character is in between two characters

3.3 Multiple characters

As all rules are applied in sequence, and hence the order of rules is important, all rules concerning multiple characters must precede all single character rules.

1. rule:

```
<rule>
    <from>&#x03B1;&#x03C5;</from> <!-- GREEK SMALL LETTER ALPHA &
                                       SMALL LETTER UPSILON -->
    <to>au</to>
</rule>
```

2. rule:

```
<rule>
    <from>&#x03B1;</from>         <!-- GREEK SMALL LETTER ALPHA -->
    <to>a</to>
</rule>
```

If you switch the order of the rules in the example above, every single "alpha" would be transliterated first and the digraph pattern will never match.

4 Building a development version

Your new transliteration table has to be converted to a Perl data structure and stored in *xml/tables.dump* in order to be put to use and tested as a development version of *Lingua::Translit*.

`xml2dump.pl` is a tool that processes XML transliteration table definitions and converts them to Perl data structures. Normally, all stable transliteration tables are processed once and stored in *xml/tables.dump* and included in the Lingua::Translit::Tables module at build time.

4.1 Using xml2dump.pl

To accomplish this task the `xml2dump.pl` tool comes in handy:

```
alinke$ ./xml2dump.pl -v -o tables.dump mytable.xml
Parsing mytable.xml... (MyTable: rules=2, contexts=1)
1 transliteration table(s) dumped to tables.dump.
```

It reads an XML definition, processes it and dumps the resulting data structure to a given file (`-o` switch).

Your transliteration table is now ready to be included by Lingua::Translit::Tables so it can be tested and evaluated.

4.2 Building a temporary Lingua::Translit

Use the standard toolchain to build a temporary development version of *Lingua::Translit* which contains nothing but your new transliteration table.

```
alinke$ perl Makefile.PL && make
```

Given the resulting development version, it's time to test the transliteration table for completeness and correct functionality.

5 Testing a transliteration table

To verify that your set of transliteration rules works correctly, you need to make some tests using your favorite Perl test framework. For an easy and complete example that utilizes the *Test::More* framework, have a look at *t/11_tr_Common_DEU.t*.

Lingua::Translit comes with a ready to use test template that you could use as a starting point and suite it to your transliterations specific needs. It is located at *t/xx_tr_template.t.pl* (and provided in appendix B) - to follow *Lingua::Translit*'s naming convention, rename it to *NN_tr_NAME.t*.

5.1 Hints on what to test

- If your transliteration is straight forward (only "1:1" mappings), just test a small text and have a look at the result. At best, everything is correct and you are ready.
- If the transliteration is reversible, you should check if both directions are transliterated correctly.
- All the context-sensitive and multi-character transliterations should be tested explicitly, to assure, that the error-prone mappings also work as expected.

5.2 Running the Tests

While testing it is convenient to define the environment variable `PERL5LIB` (have a look at `perlrun(1)`) so that the Perl interpreter knows where your development version of *Lingua::Translit* is located. The following example session assumes that you are using `bash(1)` or a similar shell:

```
alinke$ export PERL5LIB="blib/lib"
alinke$ perl t/66_tr_mytest.t
1..2
ok 1 - MyTable: not reversible
ok 2 - MyTable: transliteration
```

If all tests work as expected and hence your transliteration table is ready for usage, clean up your shell's environment and prepare to integrate your table into the existing set of transliteration tables:

```
alinke$ unset PERL5LIB
```

6 Integrating a new table

Change to the *xml/* directory and let `make(1)` call `xml2dump.pl` in order to build a data structure ("*tables.dump*") from all available XML transliteration tables, including yours:

```
alinke$ make all-tables
```

Now, clean up the old files from the development version you used to write your tests. Change into the source directory's root and run

```
alinke$ make distclean && perl Makefile.PL && make
```

The result is a complete version of *Lingua::Translit* that contains all upstream tables, as well as your own addition.

```
alinke$ make test
```

. . . assures everything is alright and ready for installation or packaging. Congratulations!

7 Contributing your table

If you like to contribute your transliteration table under the terms of the GPL/Artistic License, it can be included in the official upstream version. To accomplish this, create a patch of your changes and send it along with a description and comments to *perl@lingua-systems.com*.

A template.xml

Shortend, have a look at *xml/template.xml* for the full template.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE translit SYSTEM "translit.dtd">

<!--
  Transliteration definitions for XXX.

  Copyright 200x ... <...>
-->

<translit>
    <name></name>
    <desc></desc>
    <reverse></reverse>

    <rules>
        <!-- Without context... -->

        <!-- ...using ASCII characters -->
        <rule>
            <from>X</from>
            <to>Y</to>
        </rule>

        <!-- ...using Unicode codepoints (hexadecimal) -->
        <rule>
            <from>&#x00E4;</from>
            <to>ae</to>
        </rule>

        <!-- ...using Unicode codepoints (decimal) -->
        <rule>
            <from>&#228;</from>
            <to>ae</to>
        </rule>

        <!-- With context... -->

        <!-- ...specified as after, before or both (=between) -->
        <rule>
            <from>A</from>
            <to>B</to>
            <context>
                <after>x</after>
                <before>y</before>
            </context>
        </rule>
    </rules>
</translit>
```

B template.t

This template is located at *t/xx_tr_template.t.pl*.

```perl
use strict;
use Test::More tests => 3;  # number of tests

my $name        =   "";     # transliterations name
my $reversible  =   0;      # is the transliteration reversible?

my $input       =   "";     # short corpus...
my $output_ok   =   "";     # ...its correct transliteration

my $context     =   "";     # context-sensitive example
my $context_ok  =   "";     # ...its correct transliteration

use Lingua::Translit;

my $tr = new Lingua::Translit($name);


my $output = $tr->translit($input);

# 1
is($tr->can_reverse(), $reversible, "$name: reversibility");

# 2
is($output, $output_ok, "$name: transliteration");

$output = $tr->translit($context);

# 3
is($output, $context_ok, "$name: transliteration " .
                         "(context-sensitive)");

# vim: sts=4 sw=4 enc=utf-8 ai et
```

C References

→ Lingua-Systems' *Lingua::Translit* website,
http://www.lingua-systems.com/transliteration/Lingua-Translit-Perl-module/

→ *Lingua::Translit* on CPAN,
http://search.cpan.org/dist/Lingua-Translit/

→ ISO 9 Standard (1995) "Transliteration of Cyrillic characters into Latin characters",
http://www.iso.org/iso/iso_catalogue.htm

→ ISO 843 Standard (1997) "Conversion of Greek characters into Latin characters",
http://www.iso.org/iso/iso_catalogue.htm

→ DIN 1460 Standard (1982) "Conversion of cyrillic alphabets of slavic languages",
http://www.nabd.din.de/

→ DIN 31634 Standard (1982) "Conversion of the Greek alphabet",
http://www.nabd.din.de/

→ Streamlined Sytem (1995) "Romanization of Bulgarian",
http://members.lycos.co.uk/rre/Streamlined.html