NAME

Locale::Maketext::Cookbook - recipes for using Locale::Maketext

INTRODUCTION

This is a work in progress. Not much progress by now :-)

ONESIDED LEXICONS

Adapted from a suggestion by Dan Muey

It may be common (for example at your main lexicon) that the hash keys and values coincide. Like that

```
q{Hello, tell me your name}
=> q{Hello, tell me your name}
```

It would be nice to just write:

```
q{Hello, tell me your name} => ''
```

and have this magically inflated to the first form. Among the advantages of such representation, that would lead to smaller files, less prone to mistyping or mispasting, and handy to someone translating it which can simply copy the main lexicon and enter the translation instead of having to remove the value first.

That can be achieved by overriding `init` in your class and working on the main lexicon with code like that:

```
package My::I18N;
...

sub init {
my $lh = shift; # a newborn handle
$lh->SUPER::init();
inflate_lexicon(\%My::I18N::en::Lexicon);
return;
}

sub inflate_lexicon {
my $lex = shift;
while (my ($k, $v) = each %$lex) {
$v = $k if !defined $v || $v eq '';
}
}
```

Here we are assuming `My::I18N::en` to own the main lexicon.

There are some downsides here: the size economy will not stand at runtime after this `init()` runs. But it should not be that critical, since if you don't have space for that, you won't have space for any other language besides the main one as well. You could do that too with ties, expanding the value at lookup time which should be more time expensive as an option.

DECIMAL PLACES IN NUMBER FORMATTING

After CPAN RT #36136 (https://rt.cpan.org/Ticket/Display.html?id=36136)

The documentation of Locale::Maketext advises that the standard bracket method `numf` is limited and that you must override that for better results. It even suggests the use of Number::Format.

One such defect of standard `numf` is to not be able to use a certain decimal precision. For example,

```
$lh->maketext('pi is [numf,_1]', 355/113);
```

outputs

```
pi is 3.14159292035398
```

Since pi X 355/116 is only accurate to 6 decimal places, you would want to say:

```
$lh->maketext('pi is [numf,_1,6]', 355/113);
```

and get ‘‘pi is 3.141592’’.

One solution for that could use `Number::Format` like that:

```
package Wuu;

use base qw(Locale::Maketext);

use Number::Format;

# can be overridden according to language conventions
sub _numf_params {
return (
-thousands_sep => '.',
-decimal_point => ',',
-decimal_digits => 2,
);
}

# builds a Number::Format
sub _numf_formatter {
my ($lh, $scale) = @_;
my @params = $lh->_numf_params;
if ($scale) { # use explicit scale rather than default
push @params, (-decimal_digits => $scale);
}
return Number::Format->new(@params);
}

sub numf {
my ($lh, $n, $scale) = @_;
# get the (cached) formatter
my $nf = $lh->{_ _nf}{$scale} ||= $lh->_numf_formatter($scale);
# format the number itself
return $nf->format_number($n);
}

package Wuu::pt;

use base qw(Wuu);
```

and then

```
my $lh = Wuu->get_handle('pt');
$lh->maketext('A [numf,_1,3] km de distancia', 1550.2222);
```

would return ‘‘A 1.550,222 km de distância’’.

Notice that the standard utility methods of `Locale::Maketext` are irremediably limited because they could not aim to do everything that could be expected from them in different languages, cultures and applications. So extending `numf`, `quant`, and `sprintf` is natural as soon as your needs exceed what the standard ones do.