mitgliedsgebühren.pl

sECuRE, 2012-09-20

powered by LuaLaTeX

The problem

`10` We need membership fees to pay our server/domains

`20` Our members are nerds
$\Rightarrow$ They forget paying

`30` We should send reminders, but `GOTO 20`

The plan

Phase 1: Get list of members and bank account logs

Phase 2: ???

Phase 3: PROFIT

aqbanking transactions file

```
transaction {
  int  transactionId="1973"
  char date="20120703"
  value {
    char value="1200%2F100"
    char currency="EUR"
  } #value
  char transactionText="DA-GUTSCHR"
  char localBankCode="70035000"
  char localAccountNumber="544xxxyyy1"
  char remoteBankCode="36010043"
  char remoteAccountNumber="071xxxyyy2"
  char remoteName="MICHAEL STAPELBERG"
  char purpose="MITGLIEDSBEITRAG"
} #transaction
```

aqbanking transactions (2)

```
transaction {
…
char purpose="RZL-BEITRAG JUNI, DANKE", "98095%2FNONAME E.V.,"
} #transaction
```

bank transactions (3)

- ISO 646: 7 bit-encoding, mostly (!) like US-ASCII
- Every country has its own national variant

hex	char	hex	char
23	#	5D	Ö
24	$	5E	^
26	&	60	`
2F	/	7B	ä
40	§	7C	ö
5B	Ä	7D	ü
5C	Ö	7E	ß

bank transactions (4)

ISO 646 contains Umlauts, let's use them. Sometimes.

Real name	Bank account
Bubel, André-Patrick	ANDRE-PATRICK BUBEL
Schütz, Matthias	MATTHIAS SCHUETZ
Töpper, Thorsten	THORSTEN TÖPPER
Haufe, Jakob	HAUFE,JAKOB

bank transactions (5)

```
sub name_strip {
    my ($name) = @_;
    $name = lc $name;   # Ignore case
    $name =~ s/,/ /g;   # Strip commas
    $name =~ s/  */ /g; # double whitespace
    $name =~ s/ä/ae/g;  # Umschrift
    $name =~ s/ü/ue/g;
    $name =~ s/ö/oe/g;
    $name =~ s/ß/ss/g;
    $name =~ s/[^a-z -]/./gu; # Strip non-ascii
    return $name;
}
```

UTF-8 in Perl

```
# So that we can use UTF-8 in the source code
use utf8;

# So that files are always treated as UTF-8
use open ':encoding(utf8)';

# Putting UTF-8 to stdout is okay
binmode STDOUT, ":utf8";

# Yes, templates use UTF-8, too.
my $tt = Template->new({ ENCODING => 'utf8' });
open(my $fh, '>', "mailqueue/$name");
$tt->process('templates/foo', {}, $fh,
    binmode => ':utf8');

# Match Unicode glyphs, not bytes.
$name =~ s/[^a-z -]/./gu;
```

Locales in Perl

```perl
# Use the current locale.
use locale;

# Otherwise we don't have LC_NUMERIC etc.
use POSIX qw(locale_h);

# For "108,00 €" instead of "108.00 €"
# de_DE doesn't work, only de_DE.UTF-8
setlocale(LC_NUMERIC, 'de_DE.UTF-8');

# Beware: using DBIx::Abstract breaks
# setting LC_NUMERIC. Re-set afterwards.
```

Easier/reliable matching

- Each member gets a different transaction subject
- How do we generate them in the best possible way?
- Use only `A C D E H K L P T W X Y 3 4 7 9`
- Common prefix, easy formatting: `MB-PP-47-LT`
- Need some protection against typos

Typo protection (1)

- Reed-Solomon?
- Encode 3 x 5 bits (125 members), add 3 x 5 bits redundancy
- Will only match typos, not missing letters:
  fixes `MB-PP-48-LT` but not `MB-PP-4-LT`
- Recognizes two errors, fixes one error

Typo protection (2)

- Levenshtein distance (edit distance)!
- Assign subjects with maximum levenshtein distance to every other subject
- ”Catches/fixes” more errors, at any position (2 guaranteed fixes, more than two are likely)
- Much easier to implement
- How do you generate them?

Maximizing Levenshtein distance

```
my @okay = qw(A C D E H K L P T W X Y 3 4 7 9);

sub generate_num {
    join('', map { $okay[hex($_)] } split //,
        substr(sha1_hex($_[0]), 0, 6));
}

sub min_distance {
    my ($cur) = @_;
    my $min = length($cur);
    for my $other (@existing) {
        my $dist = edistance($cur, $other);
        $min = $dist if $dist < $min;
    }
    return $min;
}

$x++ while min_distance(generate_num($x)) < 5;
```