DBD::PO

Mit SQL GNU gettext
PO-Files bearbeiten

Über was ich heute spreche.

- Vom Quelltext bis zur mehrsprachigen Anwendung.
- Was kann ein Übersetzungsbüro.
- Formate
- Irrwege
- Lösungen

Am Anfang ist der Quelltext.

```
print
    'You can log out here.';

printf
    '%s books are in %s shelves.',
    5,
    2;
```

Markieren

```
print translate(
    'You can log out here.',
);

printf translate(
    '%s books are in %s shelves.',
), 5, 2;
```

Extrahieren

```
Zeile 1:
 'You can log out here.'

Zeile 2:
 '%s books are in %s shelves.'
```

Übersetzen Englisch-Deutsch

```
'You can log out here.'
=> 'Sie können sich hier abmelden.'

'%s books are in %s shelves.,
=> '%s Bücher sind in %s Regalen.'
```

So geht das nicht!

- Wir benötigen ein vernünftiges Format zum Transport zum Übersetzer.
- Man könnte alles in einer Excel-Datei speichern.
- Gibt es ein Format, welches für so etwas entworfen wurde?

PO-Files - was ist das?

- PO ist die Abkürzung für 'Portable Object'.
- GNU getext PO-Files kann man benutzen, um Programme mehrsprachig zu machen.
- Im File stehen neben dem Originaltext und der Übersetzung verschiedene Kommentare und Flags.

Das sieht dann so aus.

```
#: my_program.pl:1
msgid "You can log out here."
msgstr ""

#: my_program.pl:2
msgid "%s books are in %s shelves."
msgstr ""
```

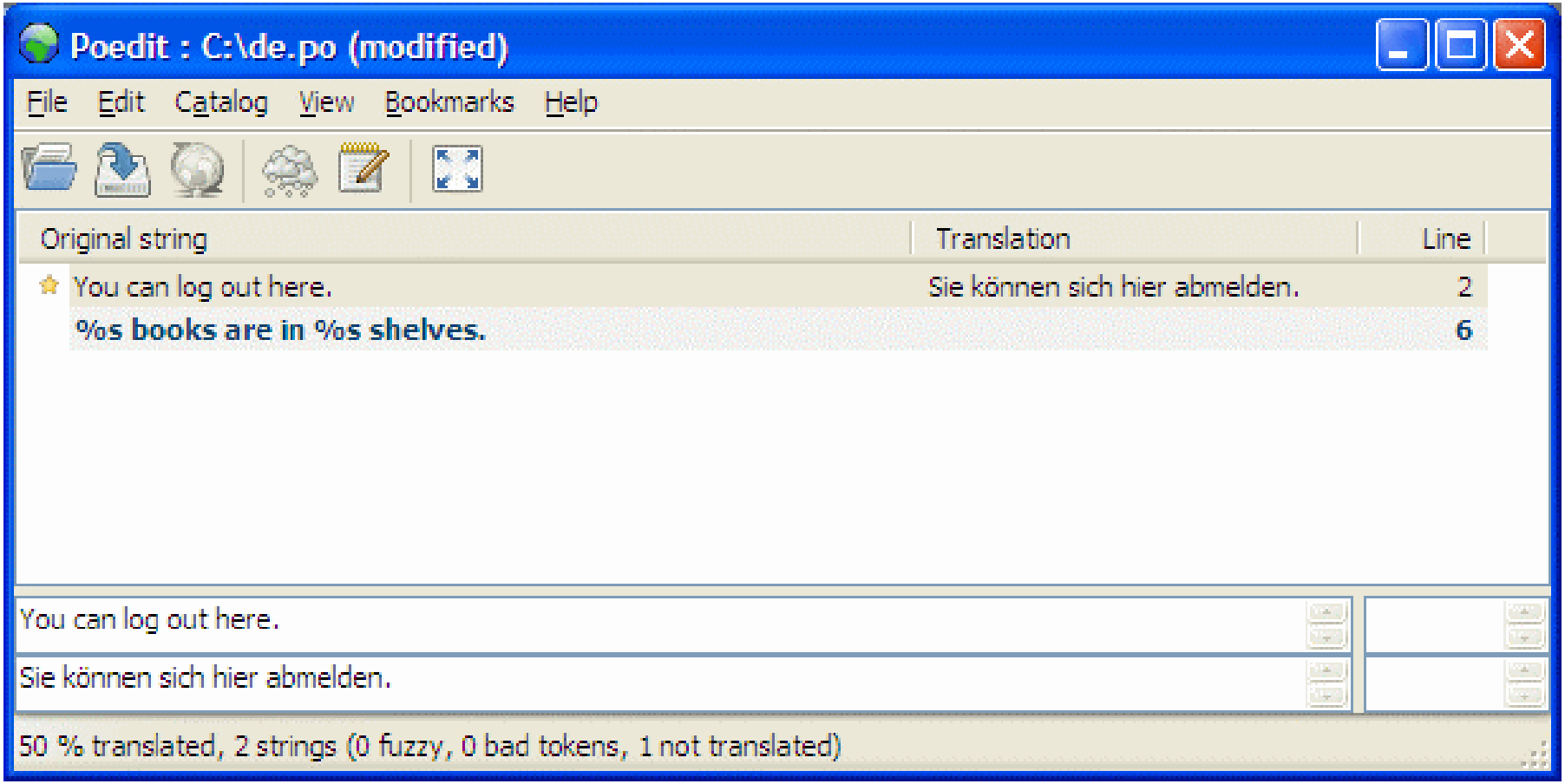
Was macht der Übersetzer damit?

Je nachdem, was man bezahlen will,

bekommt man:

- eine automatische Übersetzung,
- eine manuell nachkorrigierte,
- oder eine Handübersetzung.

Poedit
ein Übersetzungsprogramm



Das Ergebnis sieht dann so aus.

```
#: my_program.pl:1
msgid "You can log out here."
msgstr "Sie können sich hier abmelden."

#: my_program.pl:2
msgid "%s books are in %s shelves."
msgstr "%s Bücher sind in %s Regalen."
```

Platzhalter

- Übersetzt werden Textphrasen, welche auch Platzhalter enthalten können.
- Eine Variante sind die von printf bekannten %s o.ä.
- Diese Variante hat den Nachteil, dass sich die Reihenfolge der Platzhalter im Übersetzungsprozess ändern kann.
- Bei Platzhaltern wie %1, %2, usw. passiert das nicht.

Das folgende Beispiel ist zwar nur deutsch,
zeigt aber anschaulich das Problem.

5 Bücher sind in 2 Regalen.
In 2 Regalen sind 5 Bücher.

Beide Sätze sind inhaltlich gleich und können mit Platzhalten versehen werden.

%1 Bücher sind in %2 Regalen.
In %2 Regalen sind %1 Bücher.

Quantifizierung
und Zahlendarstellung

Wir benötigen dafür kein Regal.

Wir benötigen dafür 0,57 Regale.

Wir benötigen dafür 1 Regal.

Wir benötigen dafür 2 Regale.

...

Mehr dazu später.

Einzahl- bis Mehrzahl

Es ist kein Buch.

Es ist 1 Buch.

Es sind 2 Bücher.

Es sind 3 Bücher.

...

- Sowohl Poedit, als auch Locale::Maketext unterstützen Pluralformen nicht.
- Das ist nicht wirklich ein Problem.
- Man hinterlegt im Programm einfach mehrere Textphrasen.

Wie viele PO-Files braucht man?

- Im einfachsten Fall existiert 1 PO-File je zusätzliche Sprache.
- Im Beispiel ist das 'de.po' für Englisch/Deutsch.
- Man kann PO-Files für weitere Sprachvarianten, wie Dialekte, verschiedene Kontexte oder auch anwendungsspezifische Files benutzen, mit denen man dann die jeweilige Standardübersetzung ignoriert.
- Das kann man sich wie Vererbung vorstellen.

Sprachvarianten/Dialekte

- So kennt man britisches und amerikanisches Englisch.
- Und zwischen Deutschland und Österreich ist nicht alles gleich.
- In den ergänzenden PO-Files findet man jeweils nur die Abweichungen zur jeweiligen Basissprache vor.

Kontext

'Sie können sich hier abmelden.'

- Was bedeutet das?
- Meldet man sich vom Büchereiprogramm ab oder ist man danach nicht mehr Kunde der Bibliothek?
- Im englischen Text ist das klarer.
- So unterschiedlich sind Sprachen.

Warum SQL?

- Die Idee kam mir, weil ich ein Projekt kenne, in dem die Übersetzungen in einer Datenbank gespeichert sind.
- Es ist recht einfach, so eine Datenbank inhaltlich zu analysieren.
- Es ist genau so einfach, PO-Files daraus zu erzeugen und umgekehrt.
- Somit ist auch die Übersetzung über PO-Files möglich, ohne dass der Übersetzer Datenbankzugang hat.

Anwendungen wachsen

- PO-Files können recht groß und damit unübersichtlich werden.
- Mit einem einfachen Texteditor kann man sie zwar bearbeiten, jedoch sind sie auch schnell kaputt.
- Ein PO-Editor ist recht praktisch aber typisch für Übersetzer gedacht und erfüllt somit mehr dessen Anforderungen.
- Somit war es naheliegend einen Datenbanktreiber für PO-Files zu erstellen.

Irrwege

Im Original sind keine englischen Texte vorhanden.

- Manch einer hat im Quelltext der Anwendung keine englischen Texte, sondern Schlüsselworte gespeichert.

```
print translate('help');
```

- Bei recht langen Texten mag das vorteilhaft erscheinen.
- Der Übersetzer übersetzt 'help' in 'Hilfe' und nicht wie gewünscht in den langen Text, der eigentlich ausgegeben werden soll.
- Außerdem ist nun für den Text 'help' = 'Hilfe' kein Schlüssel mehr da.

Ausweg

- Diese Schlüsselworte kann man wie eine virtuelle Sprache ansehen.
- Wenn man die Übersetzung angeht, hat man dann aber:

  Virtuell -> Englisch
  Virtuell -> Deutsch
  Virtuell -> ...

- Damit kann ein Übersetzer nicht arbeiten.

- Beim SQL kennt man join.

- Also joint man:

  Englisch -> Virtuell <- Deutsch
  Englisch -> Virtuell <- ...

- Somit kann man die üblichen PO-Files erstellen, welche man nun auch übersetzten kann.

Wenn der Übersetzer kein Englisch kann.

- Es gibt PO-Files für:

  Englisch -> Deutsch
  Englisch -> Polnisch

- Es fehlt:

  Englisch -> Russisch

Der Übersetzer kann Polnisch in Russisch übersetzen.

- Also joint man:

  Polnisch -> Englisch <- Russisch

- Somit kann man ein polnisch-russisches PO-File erstellen.
- Der Übersetzer erstellt die fehlenden russischen Texte.
- Aus dem übersetzten polnisch-russischen PO-File schreibt man das englisch-russische File zurück.

Wie funktioniert DBD::PO?

- Man muss nur wissen, dass das Verzeichnis die Datenbank ist und das File die Tabelle.
- Jeder Eintrag im PO-File widerspiegelt einen zu übersetzenden Text.
- Damit ist klar, wie die Tabelle strukturiert sein muss.
- Sie hat so viele Spalten, wie es Möglichkeiten gibt, einen Eintrag zu beschreiben.

Das sind:

- msgid VARCHAR
  Text, der übersetzt werden soll
  (einzeilig/mehrzeilig)
  Nur im Header ist dieser Text leer.
- msgstr VARCHAR
  Der übersetzte Text (einzeilig/mehrzeilig)
  Der Text ist leer, wenn keine Übersetzung
  existiert.
- comment VARCHAR
  Übersetzer-Kommentar (einzeilig/mehrzeilig)
- automatic VARCHAR
  Automatischer Kommentar (einzeilig/mehrzeilig)

- reference VARCHAR
  Ein Kommentar, welcher beschreibt, wo der Text her ist (mehrzeilig)
- msgctxt VARCHAR
  Angaben zum Kontext als Text (einzeilig/mehrzeilig).
- fuzzy INTEGER
  Die Übersetzung ist abgeschlossen oder nicht.
- obsolete INTEGER
  Übersetzung wird benutzt oder nicht.

Dazu kommen noch Format-Flags, welche am Beispiel c-format beschrieben werden:

- c_format INTEGER (c-format, no-c-format)
- Format-Flags können nicht gesetzt, gesetzt oder negativ gesetzt werden.

Dazu kommen noch die Plural-Varianten:

- msgid_plural VARCHAR
  Text in der Plural-Variante, der übersetzt werden soll (einzeilig/mehrzeilig)
- msgstr_0 ... msgstr_3 VARCHAR
  Der übersetzte Text (einzeilig/mehrzeilig) in verschiedenen Plural-Varianten.

Und letztlich noch die als alt markierten Felder:

- previous_msgctxt VARCHAR
  Bisherige Angaben zum Kontext als Text (einzeilig/mehrzeilig).
- previous_msgid VARCHAR
  Bisheriger Text, der übersetzt werden soll (einzeilig/mehrzeilig)
- previous_msgid_plural VARCHAR
  Bisheriger Text in der Plural-Variante, der übersetzt werden soll (einzeilig/mehrzeilig)

Es gibt also einen Header im PO-File.

- Im Header steht als wichtigste Information der Zeichensatz.
- Somit ist immer klar, wie die Datei gelesen werden muss.

Ansonsten steht da noch:

```
Schlüsselname        | möglicher Inhalt
---------------------+---------------------------------
Project-Id-Version   | 'Project name'          # oder nichts
POT-Creation-Date    | '2008-05-31T21:02:14Z' # oder nichts
PO-Revision-Date     | '2008-07-12T07:32:56Z' # oder nichts
Last-Translator-Name | 'Steffen Winkler'       # oder nichts
Last-Translator-Mail | 'steffenw@example.org' # oder nichts
Language-Team-Name   | 'MyTeam'                # oder nichts
Language-Team-Mail   | 'cpan@example.org'      # oder nichts
```

und auch das, was DBD::PO selbst setzt:

```
Schlüsselname             | möglicher Inhalt
--------------------------+----------------------------
Content-Type              | 'text/plain'
charset                   | $po_charset || 'iso-8859-1'
Content-Transfer-Encoding | '8bit'
```

und wenn man die plural-Formen benutzt:

```
Schlüsselname | möglicher Inhalt
--------------+-----------------------------
Plural-Forms  | # Beispiel: Deutsch/Englisch
              | 'nplurals=2; plural=n != 1;'
```

und man kann selbst noch individuell erweitern:

```
Schlüsselname | möglicher Inhalt
--------------+-------------------------------------
extended      | # arrayref von Paaren
              | [qw(
              |     X-Poedit-Language      German
              |     X-Poedit-Country       GERMANY
              |     X-Poedit-SourceCharset utf-8
              | )]
```

Beispiel

```
msgid ""
msgstr ""
"Project-Id-Version: \n"
"POT-Creation-Date: \n"
"PO-Revision-Date: \n"
"Last-Translator: Steffen Winkler <steffen.winkler@example.org>\n"
"Language-Team: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=utf-8\n"
"Content-Transfer-Encoding: 8bit\n"

#: my_program.pl:1
msgid "You can log out here."
msgstr "Sie können sich hier abmelden."

...
```

Schreiben eines PO-Files

Verbindung zur Datenbank herstellen:

```
my $dbh = DBI->connect(
    "DBI:PO:f_dir=/temp/;po_charset=utf-8",
    undef,
    undef,
    {
        RaiseError => 1,
        PrintError => 0,
    },
) or die 'Cannot connect: ' . DBI->errstr();
```

Tabelle erstellen:

```
$dbh->do(<<'EOT');
    CREATE TABLE
        de.po (
            msgid   VARCHAR,
            msgstr  VARCHAR
        )
EOT
```

Header schreiben:

```
my $header
    = dbh->func('build_header_msgstr');

$dbh->do(<<'EOT', undef, $header);
    INSERT INTO de.po (
        msgstr
    ) VALUES (?)
EOT
```

Daten schreiben:

```
my $text_en = 'You can log out here.';
my $text_de = 'Sie können sich hier abmelden.';

my $sth = $dbh->prepare(<<'EOT');
    INSERT INTO $table (
        msgid,
        msgstr
    ) VALUES (?, ?)
EOT

$sth->execute($text_en, $text_de);
```

Verarbeiten eines PO-Files in einer Anwendung

Im Lexikon festlegen, woher die Daten bei der Sprache Deutsch geholt werden:

```
package Example::L10N;

use base qw(Locale::Maketext);
use Locale::Maketext::Lexicon;

Locale::Maketext::Lexicon->import({
    de       => [
        Gettext => '/po_path/de.po',
    ],
    _decode => 1, # unicode mode
});
```

Language-Handle erzeugen:

```
use Example::L10N;

my $lh = Example::L10N->get_handle('de')
    or die 'What language';
```

und für Deutsch die Zahlendarstellung einstellen:

```
$lh->{numf_comma}
     = $language =~ m{\A de_}xms;
```

und damit arbeiten:

```
print $lh->maketext(
    'You can log out here.,
);

print $lh->maketext(
    '[_1] books are in [_2] shelves.',
    5,
    2,
);
```

Die Maketext-Schreibweise für Platzhalter ist nicht %1 sondern [_1].

Wie funktioniert Quantifizierung in einer Anwendung?

```
for my $quantity (0, 0.57, 1, 2) {
 print $lh->maketext(
   'We need [*,_1,shelf,shelves,no shelf] for this.',
   $quantity,
 );
}
```

Wir benötigen dafür kein Regal.

Wir benötigen dafür 0,57 Regale.

Wir benötigen dafür 1 Regal.

Wir benötigen dafür 2 Regale.

Bibliographie

- GNU gettext in Wikipedia
  http://en.wikipedia.org/wiki/Gettext
  http://www.gnu.org/software/gettext/gettext.html
- CPAN-Modul DBD::PO
  http://search.cpan.org/dist/DBD-PO/
- CPAN-Modul DBI
  http://search.cpan.org/dist/DBI/
- CPAN-Modul Locale::Maketext
  http://search.cpan.org/dist/Locale-Maketext/
- CPAN-Modul Locale::Maketext::Lexicon
  http://search.cpan.org/dist/Locale-Maketext-Lexicon/