Open-Office-Dokumente erstellen mit Perl

Ab die Post!

Um nur schnell einen Brief am Computer zu schreiben, lohnt sich der Start von Open Office kaum. Wer nicht zum Kugelschreiber greifen will, wirft den Perl-Interpreter an. Ein Modul hilft auf Open-Office-Dokumente zuzugreifen. So sind sauber formatierte Briefe in Windeseile erstellt und ausgedruckt. Michael Schilli

Open Office ist eine echte Alternative zu reinen Windows-Lösungen wie Microsoft Office. Die Installation gelingt mit wenigen Schritten und Open-Office-Dokumente lassen sich – anders als ihre proprietären Kollegen – hervorragend von anderen Programmen aus manipulieren. So ist es nicht mal nötig, Open Office zu starten.

Natürlich bietet auch Perl ein Modul für den Zugriff auf Open-Office-Daten. Das Programm Mailit (siehe Listing 1) hilft dabei, schnell und unkompliziert Briefe zu erstellen. Dazu arbeitet es mit einer Vorlage wie in Abbildung 1. In diese baut es Adressaten, Betreff, Text und das aktuelle Datum ein. Für alle Felder finden sich in der Vorlage Platzhalter:

- »[% date %]«: Datum
- »[% recipient %]«: Empfänger-Adresse
- »[% subject %]«: Betreff
- »[% text %]«: Brieftext

Mailit erzeugt aus einer reinen Textdatei wie in Abbildung 3 unter Verwendung der Vorlage einen sauber gesetzten Brief (siehe Abbildung 2). Die Textdatei folgt einem einfachen Format: Der erste Absatz enthält die Betreffzeile des späteren Briefs, alle folgenden geben den Brieftext an. Mailit generiert das Datum rechts oben und formatiert es entsprechend der eingestellten Landessprache.

Auf den Schultern von Riesen

Bei der Implementierung von Mailit kamen sage und schreibe fünf CPAN-Module zum Einsatz, deren Möglichkeiten üblicherweise weiter reichen als nur für kurze Skript-Eskapaden. Das erste ist »OpenOffice::OODoc«, es bietet eine objektorientierte Schnittstelle zu Inhalt und Struktur von Open-Office-Dokumenten. Für die Aufgabe von Mailit (reine Textersetzung) genügt die Unterklasse »OpenOffice::OODoc::Text«.

Der Konstruktor »new« öffnet in Zeile 24 von Mailit zuerst die angegebene Open-Office-Datei, die, wie Abbildung 1 zeigt, ein Template-Dokument im gewünschten Format mit den Platzhaltern enthält. Die Methode »getTextElementList()« extrahiert später eine Liste aller Textelemente im Dokument. Die zurückgegebenen Werte sind Referenzen auf Objekte vom Typ »XML::XPath::Node::Element«, da »OpenOffice::OODoc« unter der Haube »XML::XPath« für die interne Darstellung der XML-basierten Open-Office-Dateien nutzt.

Um den Text des von der List-Funktion zurückgelieferten Elements »$e« zu extrahieren, ruft der Programmierer die Methode »getText()« des »OpenOffice::OODoc::Text«-Objekts auf und übergibt dann die Elementreferenz: »$doc->getText($e)«. Den so erhaltenen Text, der typischerweise einen ganzen Absatz im Open-Office-Dokument repräsentiert, untersucht Mailit auf vorkommende Platzhalter im Format »[% *xxx* %]« und ersetzt deren Werte entsprechend den Vorgaben.

Das Ergebnis schreibt das Programm anschließend mit Hilfe der »setText()«-Methode (Zeile 66) wieder zurück ins Dokument, ebenfalls unter Angabe der entsprechenden Elementreferenz: »$doc->setText($e, $text)«

Textersetzung leicht gemacht

Die eigentliche Textersetzung nimmt das zweite Modul, das mächtige Template Toolkit vor. Es ist das neue In-Modul für Webapplikationen, die von Designern gestaltete HTML-Templates mit dynamischen Daten füllen. Mailit erzeugt in Zeile 50 ein Objekt der Klasse »Template«. Der ab Zeile 52 definierte Hash »%vars« ordnet den Platzhaltern im Dokument ihre dynamisch zugewiesenen

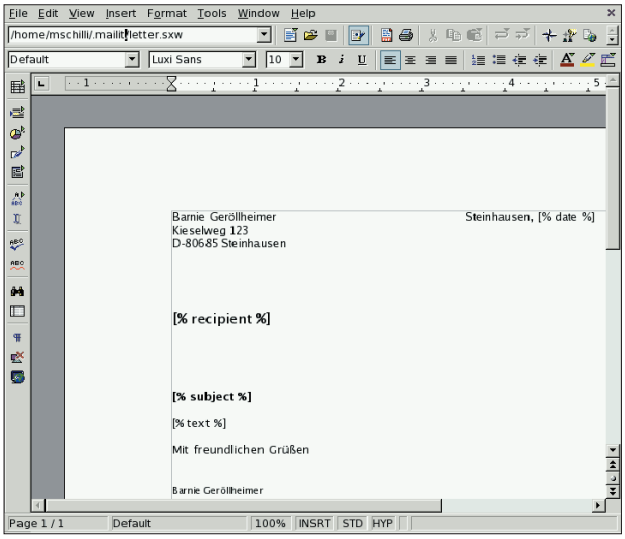

Abbildung 1: Das Office-Dokument »letter.sxw« enthält Platzhalter für dynamisch eingesetzte Textpassagen. Mailit ersetzt sie durch die eigentlichen Daten.

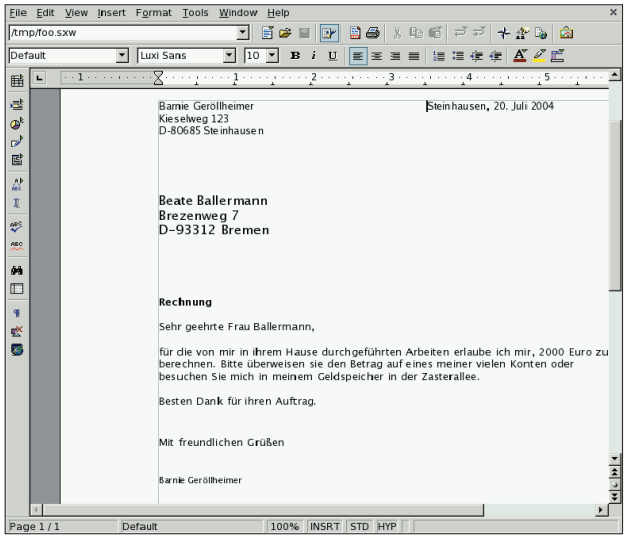

Abbildung 2: Mailit hat die Textpassage aus Abbildung 2 sowie Adressaten und Datum in die Vorlage aus Abbildung 1 eingebaut.

Werte zu. Die danach aufgerufene »process()«-Methode des »Template«-Moduls übernimmt in Mailit drei Parameter:

- Eine Referenz auf den Template-String,
- eine Referenz auf den Hash »%vars«,
- eine Referenz auf eine Funktion, der »process()« nach erfolgreicher Textersetzung den Ergebnisstring übergibt.

Letzterer ist optional, eignet sich aber in Mailit gut dafür, den bearbeiteten Text gleich per »setText()« an das Open-Office-Dokument weiterzureichen. Das so modifizierte Dokument landet in Zeile 76 per »save« in einer neuen temporären Datei, die das Modul »File::Temp« in Zeile 69 angelegt hat.

»File::Temp« ist der Cadillac unter den Tempfile-Modulen. Die Stärke dieser Module ist es, temporäre Dateien anzulegen, ohne mit bereits bestehenden zu kollidieren. Der Programmierer wählt, in welchem Verzeichnis die Datei landen soll (»DIR => '/tmp'«), welche Endung sie aufweist (»SUFFIX => '.sxw'«) und nach welcher Vorlage das Modul den Namen generiert. Der Parameter »TEMPLATE => 'ooXXXXX'« gibt an, dass nach einem einleitenden »oo« fünf zufällige Zeichen stehen sollen. Der komplette Name einer Temp-Datei sieht etwa so aus: »/tmp/oo2hkss.sxw«

Der »UNLINK«-Parameter des »File::Temp«-Konstruktors bestimmt, ob das Modul die Datei wegputzt, wenn das zugehörige Objekt erlischt. Das zurückgelieferte Handle lässt sich als File-Handle verwenden und expandiert innerhalb eines Strings (»"$oo_output"«) zum Namen der temporären Datei.

Das vierte Modul – »Date::Calc« – hilft Mailit das heutige Datum zu bestimmen und es landestypisch ins Format »*XX. Monat Jahr*« umzuwandeln. Es setzt zunächst die Locale mit »Language(Decode_Language("Deutsch"))« und ruft weiter unten »Month_to_Text()« auf, um die von der Funktion »Today()« zurückgegebene Monatsnummer in den deutschen Namen umzuwandeln.

Für die Bestimmung der mehrzeiligen Empfängeradresse, die den Platzhalter »[% recipient %]« im Dokument ersetzt, zieht Mailit eine Adressdatenbank heran. Eine der wichtigen Fragen beim Programmieren von Mailit war es daher, welches Format sich am besten für eine einfache Adressdatenbank eignet. Die exzessiven Triangel von XML führen bei einem menschlichen Leser schnell zu dreieckigen Augen und Kopfschmerzen.

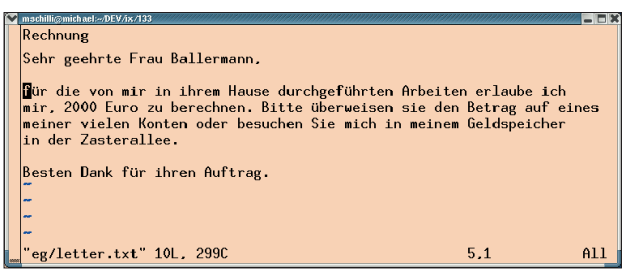

Abbildung 3: Die Textversion des Briefs im Editor Vi. Der erste Absatz gibt den Betreff an, der Rest den Brieftext.

Für Menschen lesbare Adressenbank

Das von Brian Ingerson entwickelte YAML (Yaml Ain't Markup Language) lässt sich hingegen nicht nur leicht parsen, es schmeichelt auch dem Auge. Eine Adressdatenbank, die ihre Datensätze per Kürzel indiziert und ihnen Werte für Name, Straße und Wohnort zuweist, sieht in YAML so aus:

```
otto:
  - Otto Ollenhauer
  - Olle Straße 123
  - D-82922 Ottobrunn

bea:
  - Beate Ballermann
  - Brezenweg 7
  - D-93312 Bremen
```

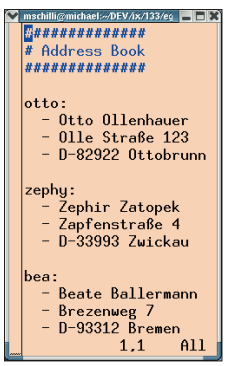

Abbildung 4: Das Adressbuch für Mailit im YAML-Format. Die YAML-Konfigurationsdatei hat ein sehr einfaches Format und ist übersichtlicher als ein entsprechendes XML-Dokument.

Reicht man den Namen der Datei an die »LoadFile()«-Funktion weiter, gibt diese eine Referenz auf einen Hash zurück, der die Kürzel als Schlüssel und die Einträge als Referenzen auf Arrays enthält:

```
{
'bea' => [
  'Beate Ballermann',
  'Brezenweg 7',
  'D-93312 Bremen'
],
'otto' => [
  'Otto Ollenhauer',
  'Olle Straße 123',
  'D-82922 Ottobrunn'
], ...
  }
```

YAML kann aber noch viel mehr. Es handelt sich – wie der Name schon andeutet – tatsächlich nicht um eine Markup-Sprache, sondern um einen vielseitigen Daten-Serialisierer. Er verwandelt Perls beliebig tief verschachtelte Core-Datenstrukturen in leicht lesbaren Ascii-Text und importiert sie anschließend wieder zurück nach Perl.

Mailit nimmt den Brief entweder als Dateinamen entgegen oder erwartet ihn auf Stdin: »mailit brief.txt« und »cat brief.txt | mailit« funktionieren ebenfalls, da Zeile 29 mit Perls magischer Eingaberaute arbeitet. Der reguläre Ausdruck in Zeile 33 trennt den ersten Absatz vom Rest des Briefs und legt die Bereiche in »$subject« und »$body« ab.

Und ab geht die Post

Die ab Zeile 82 definierte »pick()«-Funktion nimmt eine Reihe von Adressaten entgegen, präsentiert sie dem Benutzer in Form einer nummerierten Liste und lässt ihn eine Adresse auswählen. Ein typischer Ablauf sieht folgendermaßen aus:

```
mailit letter.txt
[1] bea
[2] otto
[3] zephy
Recipient [1]> 1
Preparing letter for Beate Ballermann
Printing /tmp/ooGd8H3.sxw
```

Um den fertigen Brief schließlich zu drucken, ruft das Programm in Zeile 71 Open Office mit dem Parameter »-p« auf. Er verhindert einen langwierigen Start der grafischen Oberfläche und schickt stattdessen die SXW-Datei an den Standard-Drucker. Fertig! *(mwe)* ■

Infos

[1] Listings zu diesem Artikel: [ftp://ftp.linux-magazin.de/pub/listings/magazin/2004/10/Perl]
[2] Open Office: [http://openoffice.org]

Der Autor

Michael Schilli arbeitet als Software-Entwickler für AOL/Netscape in Mountain View, Kalifornien. Er hat „Goto Perl 5" (deutsch) und „Perl Power" (englisch) für Addison-Wesley geschrieben und ist unter [mschilli@perlmeister.com] zu erreichen. Seine Homepage heißt [http://perlmeister.com].

Listing 1: Mailit

```
001 #!/usr/bin/perl
002 ##########################################
003 # mailit -- Print letters with OpenOffice
004 # Mike Schilli, 2004 (m@perlmeister.com)
005 ##########################################
006 use warnings;
007 use strict;
008
009 my $CFG_DIR = "$ENV{HOME}/.mailit";
010 my $OO_TEMPLATE   = "$CFG_DIR/letter.sxw";
011 my $ADDR_YML_FILE = "$CFG_DIR/addr.yml";
012 my $OO_EXE = "$ENV{HOME}/ooffice/soffice";
013
014 use OpenOffice::OODoc;
015 use Template;
016 use YAML qw(LoadFile);
017 use File::Temp;
018 use Date::Calc qw(Language Decode_Language
019                   Today Month_to_Text);
020
021 Language(Decode_Language("Deutsch"));
022 my ($year,$month,$day) = Today();
023
024 my $doc = OpenOffice::OODoc::Text->new(
025     file => $OO_TEMPLATE,
026 );
027
028    # Read from STDIN or file given
029 my $data = join '', <>;
030
031    # Split subject and body
032 my($subject, $body) =
033              ($data =~ /(.*?)\n\n(.*)/s);
034
035    # Remove superfluous blanks
036 my $text;
037 for my $paragraph (split /\n\n/, $body) {
038     $paragraph =~ s/\n/ /g;
039     $text .= "$paragraph\n\n";
040 }
041
042 my $yml = LoadFile($ADDR_YML_FILE);
043 my $nick = pick("Recipient", [keys %$yml]);
044
045 my $recipient = $yml->{$nick};
046
047 print "Preparing letter for ",
048       $recipient->[0], "\n";
049
050 my $template = Template->new();
051
052 my %vars = (
053     recipient => join("\n", @$recipient),
054     subject   => $subject,
055     text      => $text,
056     date      => sprintf("%d. %s %d",
057       $day, Month_to_Text($month), $year),
058 );
059
060 for my $e ($doc->getTextElementList()) {
061
062     my $text_element = $doc->getText($e);
063
064     $template->process(\$text_element,
065         \%vars,
066         sub { $doc->setText($e, $_[0]); });
067 }
068
069 my $oo_output = File::Temp->new(
070     TEMPLATE => 'ooXXXXX',
071     DIR      => '/tmp',
072     SUFFIX   => '.sxw',
073     UNLINK   => 1,
074 );
075
076 $doc->save($oo_output->filename);
077
078 print "Printing $oo_output\n";
079 system("$OO_EXE -p $oo_output");
080
081 ##########################################
082 sub pick {
083 ##########################################
084     my ($prompt, $options) = @_;
085
086     my $count = 0;
087     my %files = ();
088
089     foreach (@$options) {
090         print STDERR "[",
091               ++$count, "] $_\n";
092         $files{$count} = $_;
093     }
094
095     print STDERR "$prompt [1]> ";
096     my $input = <STDIN>;
097     chomp($input);
098
099     $input = 1 unless length($input);
100     return "$files{$input}";
101 }
```