Don't write it, stick it!

Independent Label

OpenOffice offers a selection of preconfigured formats for users who need to print their own self-adhesive labels. Perl feeds the address data to the document. *By Mike Schilli*

Even though you might enjoy writing greeting cards by hand and mailing them, you might consider saving yourself the trouble of writing the addresses on the envelopes and use sticky labels instead. Practical laser printer labels on letter-sized sheets (Figure 1) cost about half a cent apiece (15 cents a sheet). Not only do they speed up the postal shipping process, they are also useful for labeling wires of electronic gadgets. The Perl script in this issue reads comma-separated text and prints it line by line on the labels. Of course, you don't have to restrict yourself to addresses. How do you like the idea of labeling the mess of cables beneath your desk to help the overworked home network administrator find the router's power supply next time something goes wrong?

Preconfigured Sizes

OpenOffice Writer already supports label formats from a variety of vendors, and in the *New* | *Labels* menu (Figure 2), you can create tabular documents to match. In the dialog in Figure 3, you just need to enter the manufacturer and product code for the label you choose, and the dimensions are guaranteed to fit. After creating these documents, you just need to add the body text and press "Print," which is admittedly somewhat easier than writing your own printer positioning program. Because OpenOffice.org stores its documents in the open ODF format, you can extract the table data from a CSV file with a Perl script before injecting the data into the document.

ZIP Archive .odt

Before your address list can be added to the label table, you need to create a template in OpenOffice and type a couple of

MIKE SCHILLI

Mike Schilli works as a software engineer with Yahoo! in Sunnyvale, California. He can be contacted at *mschilli@perlmeister.com*. Mike's homepage can be found at *http://perlmeister.com*.

test strings (Figure 4). As the `unzip` command in Figure 5 shows, the resulting document, saved as `template.odt`, comprises a ZIP archive with XML files, the most interesting of which, `content.xml`, is the text content of the document with XML markups.

Calling the script in Listing 1 by typing `oo-dumper template.odt` then reveals the document content and the markup structures for the strings entered previously by the user in the table elements.

The script pulls in the CPAN OpenOffice::OODoc module and calls its constructor `ooDocument()` with the name of the file to be investigated. Line 18 defines `"content"` as the document `member`; in other words, I'm interested in the document content, not the external headers, footers, reusable style definitions, or metadata.

Exploring XML

The `selectElements` method launches an XPath query that reveals all the XML elements inside the `office:body` tag – that is, the text document itself. Documents only contain a single body; however, OpenOffice::OODoc insists that the left-hand side of the assignment in line 21 suggests a list context, which explains the parentheses around `$element`. The return value is a reference to a specific `OpenOffice::OODoc::Element`, which also understands the methods of executing the XML parser, XML::Twig, by inheritance. This slightly quirky XML module, which I wrote about in an earlier Perl column [3], provides the `_dump()` method, which generates the text format of the XML subtree and returns it as a string.

In Figure 6, the `office:text` tag below `office:body` contains a couple of `sequence` declarations followed by `text:p`-type text. This line is a row in a table with three columns, for which elements of `draw:frame` draw the frames. These frames, in turn, contain `draw:text-box` elements with `text:p` elements where the test text (test1, test2, etc.) resides.

An XPath query like

```
//office:body/office:text/text:p
```

retrieves all the lines in the table (which contain the column frames); whereas, the table elements (three per row) are stored relative to this below:

```
.../draw:frame/draw:text-box/text:p
```

The script in Listing 2 uses the first XPath query to add as many lines to the table as needed for the address data to be printed. The second query then browses all the labels and injects the text data into them.

To open the `.odt` template, the script uses the `ooDocument()` constructor on the `ready.odt` file which was created previously in line 31 by copying `template.odt` with the `cp` function from the Sysadm::Install module.

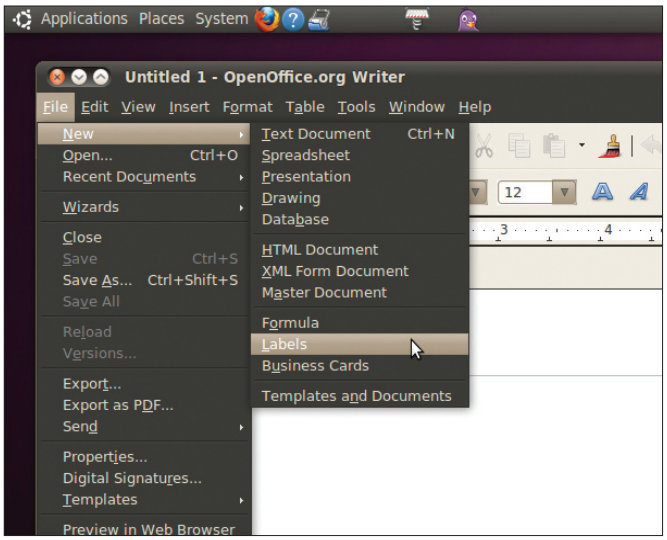

Figure 2: The Labels entry in OpenOffice Write takes you to a treasure trove of label formats.

The currently available version of OpenOffice::OODoc contains a bug that processes UTF8-encoded documents incorrectly if they contain non-ASCII characters. The

```
local_encoding => ""
```

setting in line 36 provides a temporary fix, but the value should be `"utf8"`.

Adding the Address Data

The raw data must be stored in an `address-book.csv` file (Figure 7), where the `label-writer` script can pick them up line by line with the `getline()` function from the CPAN Text::CSV_XS module.

The `addresses_scan` function opens the file with the `:encoding(utf8)` pragma,

Figure 1: Laser printer labels: 30 per sheet, 4,200 a box.

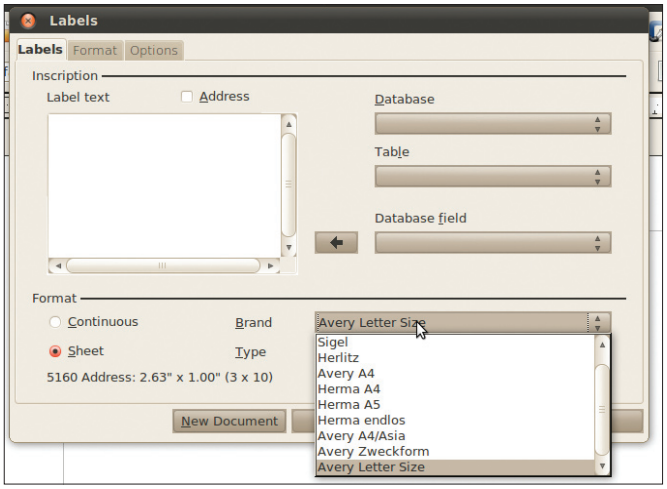

Figure 3: Besides the vendor, you can select a specific label product.

which allows Perl to parse UTF-8-encoded characters correctly in the file and set the UTF-8 flag for them in its data structures.

In each loop iteration, the `$row` variable points to an array whose elements represent the comma-separated line entries in the CSV file. To leave enough space on the left side of the label, where the text entry goes later on, the replace command within the `for` loop starting on line 99 inserts a blank in front of every line on the label. Line 104 concatenates the lines of the address to create a string with line breaks and pushes it to the end of the `@addresses` array, which the func-

LISTING 1: oo-dumper

```
01 #!/usr/local/bin/perl -w
02 ############################
03 # oo-dumper - Dump an
04 #         OpenOffice document
05 # Mike Schilli, 2011
06 # (m@perlmeister.com)
07 ############################
08 use strict;
09 use OpenOffice::OODoc;
10
11 (my $file) = @ARGV;
12
13 die "usage: $0 file"
14   unless defined $file;
15
16 my $doc = ooDocument(
17  file   => $file,
18  member => "content",
19 );
20
21 (my $element) =
22   $doc->selectElements(
23  '//office:body');
24
25 print $element->_dump();
```

LISTING 2: label-writer

```
001 #!/usr/local/bin/perl -w
002 ############################
003 # label-writer - Use
004 #   OpenOffice to write
005 #   address labels
006 # Mike Schilli, 2011
007 # (m@perlmeister.com)
008 ############################
009 use strict;
010 use OpenOffice::OODoc;
011 use Sysadm::Install
012   qw( :all );
013 use Text::CSV_XS;
014 use POSIX qw(ceil);
015
016 my $template =
017   "template.odt";
018 my $file = "ready.odt";
019 my $addr_book =
020   "address-book.csv";
021 my $labels_per_page = 30;
022
023 my @addresses =
024   addresses_scan($addr_book);
025
026 my $addtl_pages = ceil(
027  scalar @addresses /
028    $labels_per_page) - 1;
029
030 # Put template in place
031 cp $template, $file;
032
033 my $doc = ooDocument(
034  file          => $file,
035  type          => "content",
036  local_encoding => "",
037 );
038
039 # Extend document as
040 # necessary
041 my @rows =
042   $doc->selectElements(
043    '//office:body/' .
044    'office:text/text:p'
045   );
046
047 for (1 .. $addtl_pages) {
048  for my $row (@rows) {
049   $doc->replicateElement(
050    $row, "body");
051  }
052 }
053
054 # All labels, including new ones
055 my @labels =
056   $doc->selectElements(
057    '//office:body/' .
058    'office:text/text:p/' .
059    'draw:frame/' .
060    'draw:text-box/text:p'
061   );
062
063 my $addr_idx = 0;
064
065 for my $label (@labels) {
066  $doc->setStyle($label,
067   "P1");
068  $doc->setText($label,
069   $addresses[$addr_idx]);
070  $addr_idx++;
071  $addr_idx = 0
072    if $addr_idx >
073     $#addresses;
074 }
075
076 $doc->save();
077
078 ############################
079 sub addresses_scan {
080 ############################
081  my ($addr_book) = @_;
082
083  my @addresses = ();
084
085  open(my $fh,
086   "<:encoding(utf8)",
087   $addr_book)
088    or die "$addr_book: $!";
089
090  my $csv = Text::CSV_XS->new(
091   { binary => 1 })
092    or die "Cannot use CSV: "
093    . Text::CSV->error_diag();
094
095  while (my $row =
096   $csv->getline($fh)) {
097   unshift @$row, "";
098
099   for (@$row) {
100    s/^/ /;
101   }
102
103   push @addresses,
104     join("\n", @$row);
105  }
106  close $fh;
107
108  return @addresses;
109 }
```

tion passes back to the main program after completing its work.

Waste Not, Want Not!

To avoid wasting labels, the script always completely fills a letter-sized page; if needed, by repeating the addresses in the CSV file. If the address database has more than 30 entries, Listing 2 has to add additional pages at the end of the document. In this case, too, it will fill up any space left over on the page with repeat data.

Line 21 refers to the predefined number of labels per page, and line 26 calculates the required number of pages in the label document by factoring in the size of the address database. The `ceil()` function from the POSIX module rounds up to the next integer in case of fractions. The number of `$addtl_pages` (additional pages) is then one less because the template document created by the user already provides the first page.

All the table rows on the test page are lined up in the `@rows` array after the first XPath query in line 42, and for each additional page to be created, the `for` loop from line 47 to 52 iterates over these row entries, duplicates them with `replica-`

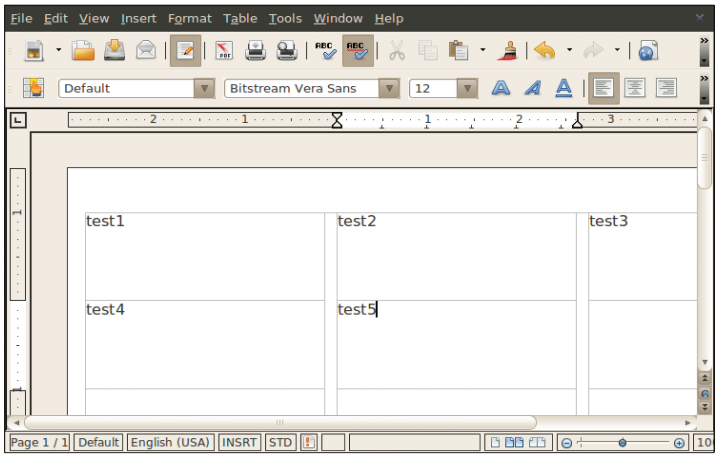

Figure 4: The user types sample texts in the text fields of the OpenOffice document.

```
$ unzip ../template.odt
Archive:  ../template.odt
 extracting: mimetype
   creating: Configurations2/statusbar/
  inflating: Configurations2/accelerator/current.xml
   creating: Configurations2/floater/
   creating: Configurations2/popupmenu/
   creating: Configurations2/progressbar/
   creating: Configurations2/menubar/
   creating: Configurations2/toolbar/
   creating: Configurations2/images/Bitmaps/
  inflating: content.xml
  inflating: manifest.rdf
  inflating: styles.xml
 extracting: meta.xml
  inflating: Thumbnails/thumbnail.png
  inflating: settings.xml
  inflating: META-INF/manifest.xml
$
```

Figure 5: A call to unzip reveals the XML files contained in the OpenOffice document.

```
| |-office:body
| | |-office:text
| | | |-text:sequence-decls
| | | | |-text:sequence-decl text:display-outline-level="0"
 text:name="Illustration"
| | | | |-text:sequence-decl text:display-outline-level="0"
 text:name="Table"
| | | | |-text:sequence-decl text:display-outline-level="0"
 text:name="Text"
| | | | |-text:sequence-decl text:display-outline-level="0"
 text:name="Drawing"
| | | |-text:p text:style-name="P1"
| | | | |-draw:frame draw:name="Frame1" draw:style-name="fr
1" draw:z-index="0" svg:height="1in" svg:width="2.6252in" t
ext:anchor-type="as-char"
| | | | | |-draw:text-box
| | | | | | |-text:p text:style-name="P1"
| | | | | | | |-PCDATA:  'test1'
| | | | |-draw:frame draw:name="Frame2" draw:style-name="fr
1" draw:z-index="1" svg:height="1in" svg:width="2.6252in" t
ext:anchor-type="as-char"
| | | | | |-draw:text-box
| | | | | | |-text:p text:style-name="P1"
| | | | | | | |-PCDATA:  'test2'
@
                                                1,1           Top
```

Figure 6: The _dump() method shows how the XML document is nested.

```
Carl Coolman,123 Chrome St.,"Cleveland, OH 44101"
Peter Prankster,456 Powell St.,"Portland, OR 97202"
Robert Rover,789 Rome Ave.,"Radcliff, KY 40160"
Wanda Wonder,888 Wolfe St.,"Wichita Falls, TX 76301"

"address-book.csv" 4L, 203C          4,1           All
```

Figure 7: Address data in CSV format.

teElement(), and uses the "body" parameter to tell the function to add duplicates at the end of the document body. The newly created lines are exact copies of the lines on the first page; that is, they still contain test data or are empty.

The second XPath query in line 56 retrieves all the table elements (three per line, including all the elements on the newly created pages) in the document and stores them in the @labels array. The for loop in line 65 then goes through and assigns the style "P1" to them. The dump in Figure 6 shows that this applies the Bitstream Vera Sans font style to the text. The subsequent call to setText() picks up the next record from the address file and stores the corresponding text string in the table element currently being processed.

The loop continuously increments the $addr_idx index variable to the address array, starting at zero, and resets it to zero again when it reaches the end of the address database to restart with the first address.

Correct Insertion

The save() method then saves the changes to ready.odt, the target file, which have only been made in volatile memory thus far to disk. When the user opens the file in OpenOffice (I tested this with version 3.2), the document looks like Figure 8.

All you need to do now is insert a page with sticky labels into the printer and select *Print* in OpenOffice Writer. To avoid wasting labels, it makes sense to print a trial page on a normal sheet of paper then hold it up to the light with the sheet of labels to check the alignment.

To find out whether the label sheet has to be face up or face down for the printer to print onto the labels and, at the same time, discover the direction in which the printer feeds the paper, put a pencil mark in one corner before printing the file. Next, you can print your test page, look at the position of the mark on the final result, and do the extremely complex geometric transformations in your head to get things right.

To install the scripts, you need the OpenOffice::OODoc, Sysadm::Install, and Text::CSV_XS modules, the latter being a speed-optimized version of the legacy Text::CSV. A CPAN shell will handle the install if these modules are not available in your choice of distribution. Next, open the OpenOffice Writer application and select the label format you need in the *Labels* dialog. You need to modify the value of 30 labels per line set in line 21 of Listing 2 to match your label format.

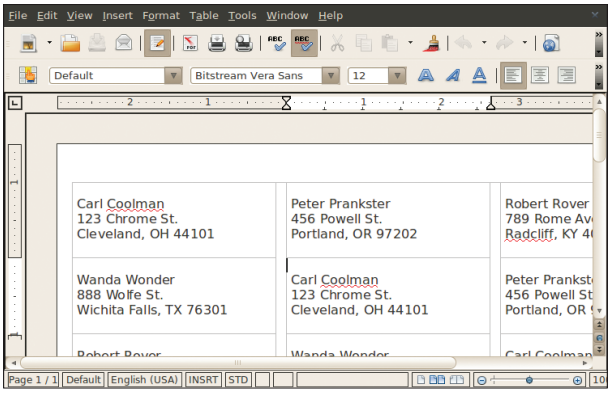

Figure 8: After running the script, the OpenOffice file ready.odt contains all the inserted addresses.

Figure 9: The printed labels.

If something goes wrong, it is a good idea to analyze the ODF file with oodumper and adjust for any deviations from the format by creating matching XPath queries.

After filling out a couple of test fields, store the results as template.odt. The label-writer script should then parse a UTF-8-encoded address-book.csv address file and create the read.odt output, which you then send to the printer.

A number of other applications come to mind for this script: labeling cables in data centers or machine numbers for asset management. Or, I might just put on my accountant's hat, slip on some oversleeves, and attach a label to all of the books in my private collection, so I know where they belong when I'm done reading them [5]. ■■■

INFO

[1] Listings for this article: *http://www.linux-magazine.com/Resources/Article-Code*

[2] "A Simple Way to Do Labels in OpenOffice Writer" by Solveig Haugland, *http://openoffice.blogs.com/openoffice/2007/06/a-simple-way-to.html*

[3] "Perl: XML Parsers" by Michael Schilli, *Linux Magazine*, September 2005, *http://www.linux-magazine.com/Issues/2005/58/SPOILED-FOR-CHOICE*

[4] "The Perl OpenDocument Connector", Jean-Marie Gouarné, *The Perl Review*, *http://www.theperlreview.com/SamplePages/ThePerlReview-v3i1.p18.pdf*

[5] "Perl: OpenOffice Access" by Michael Schilli, *Linux Magazine*, November 2004, pg. 72