Tutorial – Gerando documentos OpenOffice dinamicamente

Giovanni Floridia
dezembro de 2005

Introdução

É comum em aplicativos *web* precisem gerar relatórios ou outros documentos com algum tipo de acabamento, principalmente a capacidade de paginar o resultado. As dificuldades do HTML em lidar com documentos que devem ser impressos são bem conhecidas.
Entre as opções existentes para resolver o problema, existe a geração de arquivos PDF, pois este formato permite um melhor posicionamento dos elementos na página, garantindo também a independência da plataforma e da impressora conectada. A diagramação com tal formato, entretanto, é trabalhosa, além de ser um formato proprietário.
O presente tutorial mostra como criar dinamicamente documentos no formato OpenOffice, utilizando o próprio programa como *designer* da página: um método rápido e flexível para a geração de relatórios.

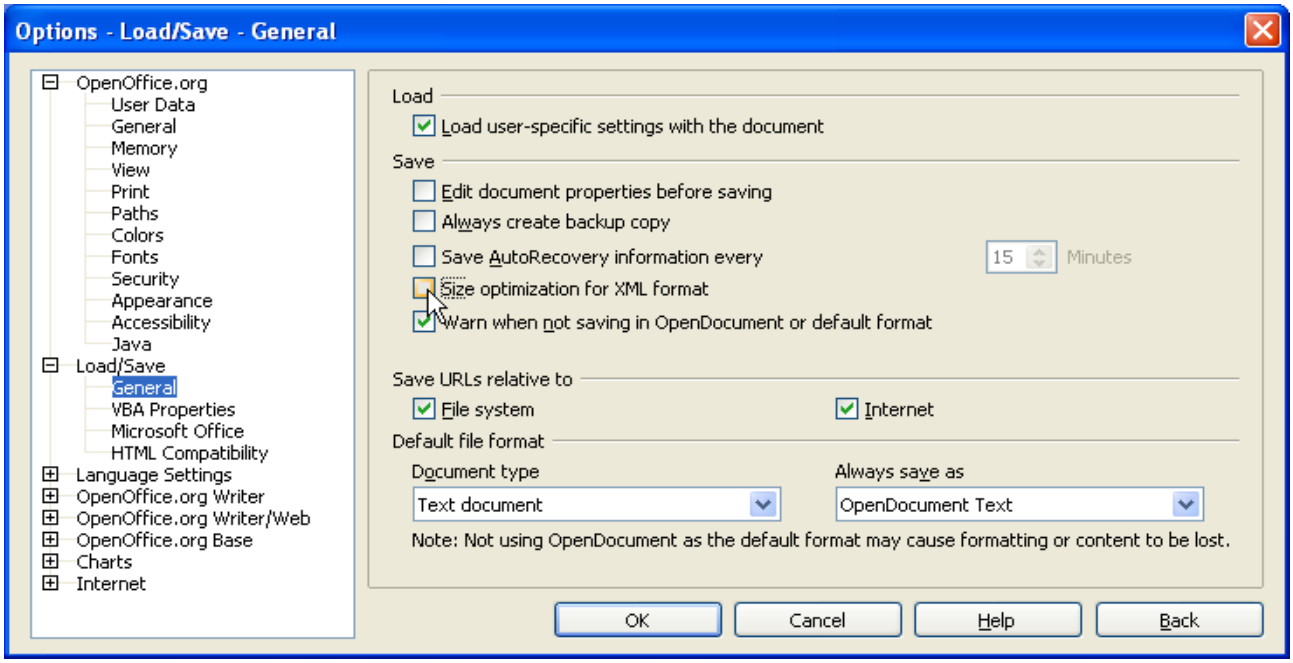
O Formato OpenDocument

A versão 2.0 do OpenOffice [1] adota um formato aberto e foi aprovado pelo consórcio OASIS [2] em 1º. de maio de 2005. O Objetivo do OASIS é criar padrões de documentos para estimular o comércio eletrônico, no sentido mais amplo possível. Vale lembrar que a adoção de padrões abertos é até mais importante que a adoção de softwares em código aberto. O formato pretende ser aberto à adoção de qualquer suíte de aplicativos de escritório e deverá ser adotado por outros programas, como o *koffice* [3] que o implementará em versão futura.
O formato é baseado em XML e sua descrição detalhada é complexa [4]. Entretanto, não é necessário conhecer as 706 páginas da especificação.
Basicamente o formato, assim como o formato anterior do OpenOffice 1.x, é uma arquivo compactado no formato zip. Quando descompactado ele apresenta a seguinte estrutura básica:

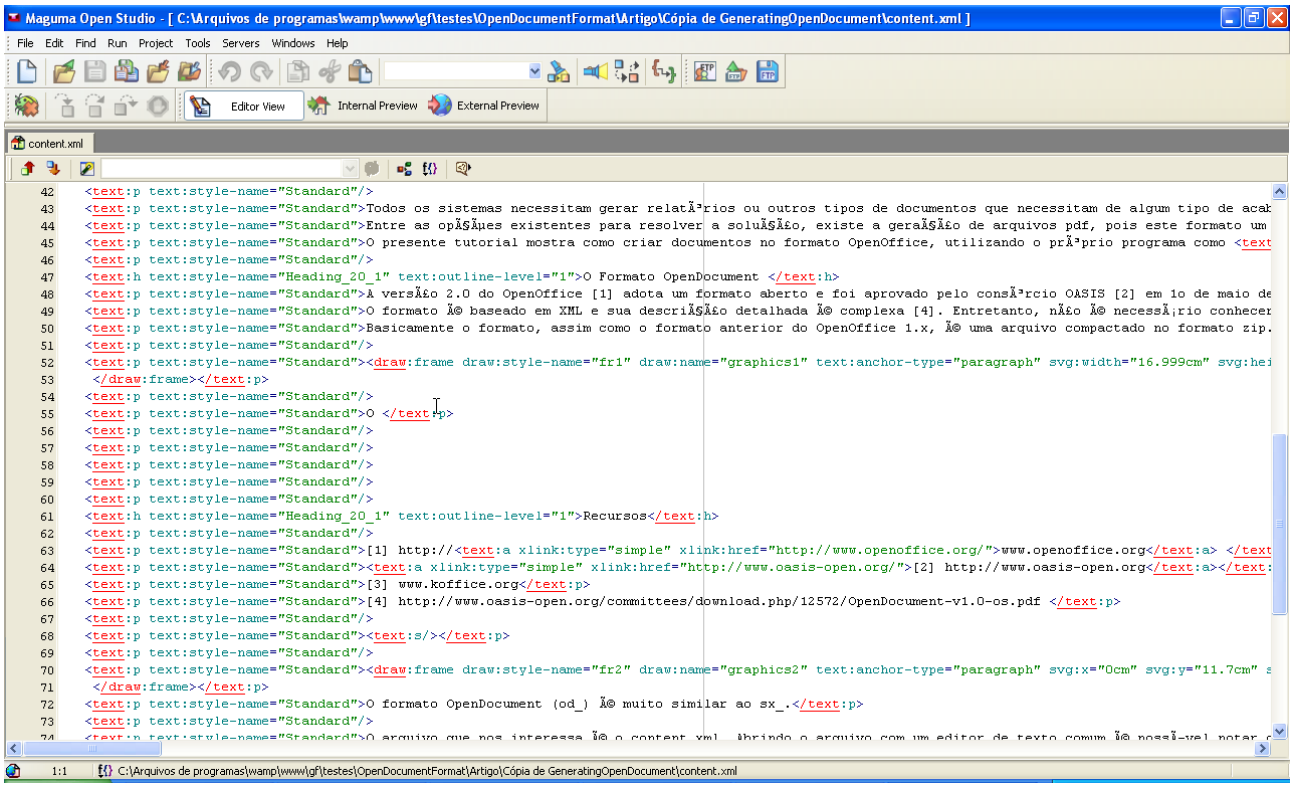
```
META_INF/
        manifest.xml
Configurations2/
Pictures/
Thumbnails/
mimetype
meta.xml
settings.xml
style.xml
content.xml
```

O arquivo que nos interessa é o content.xml. Abrindo o arquivo com um editor de texto comum é possível notar que o conteúdo está todo concentrado em uma linha, e sem as identações. Isso ocorre porque o OpenOffice procura poupar espaço. Entretanto, em nosso caso, precisaremos verificar o conteúdo dos arquivos xml gerados, portanto modificaremos um parâmetro para que isso ocorra.

Em Ferramentas | Opções | *LoadSave* | Geral desabilite a opção "*Size optimization for XML format*", ou sua equivalente, conforme a figura a seguir.



Assim, os documentos gerados serão mais legíveis, apesar de um pouco maiores. Salve o arquivo, descompacte-o e reabra o arquivo content.xml: você deverá ver um arquivo organizado.



Mesmo sem entender o arquivo, a legibilidade do XML é suficiente para que se modifique alguns parâmetros ou conteúdos e se recompacte tudo.

O Processo.

Agora já possível entender a idéia geral do processo:

1. Criamos um arquivo OpenOffice como um *template*;
2. Descompactamos em uma pasta;
3. Modificamos alguns arquivos xml, principalmente o content.xml;
4. Recompactamos tudo;
5. Enviamos ao usuário com a extensão correta.

As primeiras duas etapas devem ser feitas manualmente, usando o OpenOffice como "gerador de *templates*". As três últimas serão feitas por uma aplicação, em nosso caso utilizaremos o PHP, mas a qualquer outro tipo de linguagem poderia ser utilizada.

Nesse ponto é importante notar que é possível criar um *script* content.php que gere o content.xml desejado, capture-o, compacte-o com o resto da estrutura e o envie com a extensão adequada. Entretanto essa não é a maneira adequada de fazê-lo. Como o arquivo content.xml é gerado pelo OpenOffice, quanto mais etapas fizermos dentro do OpenOffice melhor; quanto menos modificarmos o content.xml, melhor: será mais fácil criar e manter os relatórios.

A maneira mais simples de proceder é utilizando uma ferramenta de *template*. Há dezenas de ferramentas de template, as mais conhecidas são o *Smarty* para o PHP [5], o *Cheetah*[6] para Python entre outras. Sem nenhuma perda de generalidade utilizaremos a SmartTemplate [7] de Philipp v. Criegern, por ser simples e muito rápida. Não sendo um grande usuário de templates, acredito que os exemplos iniciais apontaram-me o *Smarty* como mais longo, e quanto mais modificarmos os arquivos originais, pior é manutenção dos relatórios. Entretanto essa convicção não está fundamentada em um conhecimento das ferramentas disponíveis.

Para a compactação utilizaremos uma versão modificada da ziplib.php que é parte do projeto Phpwiki [8].

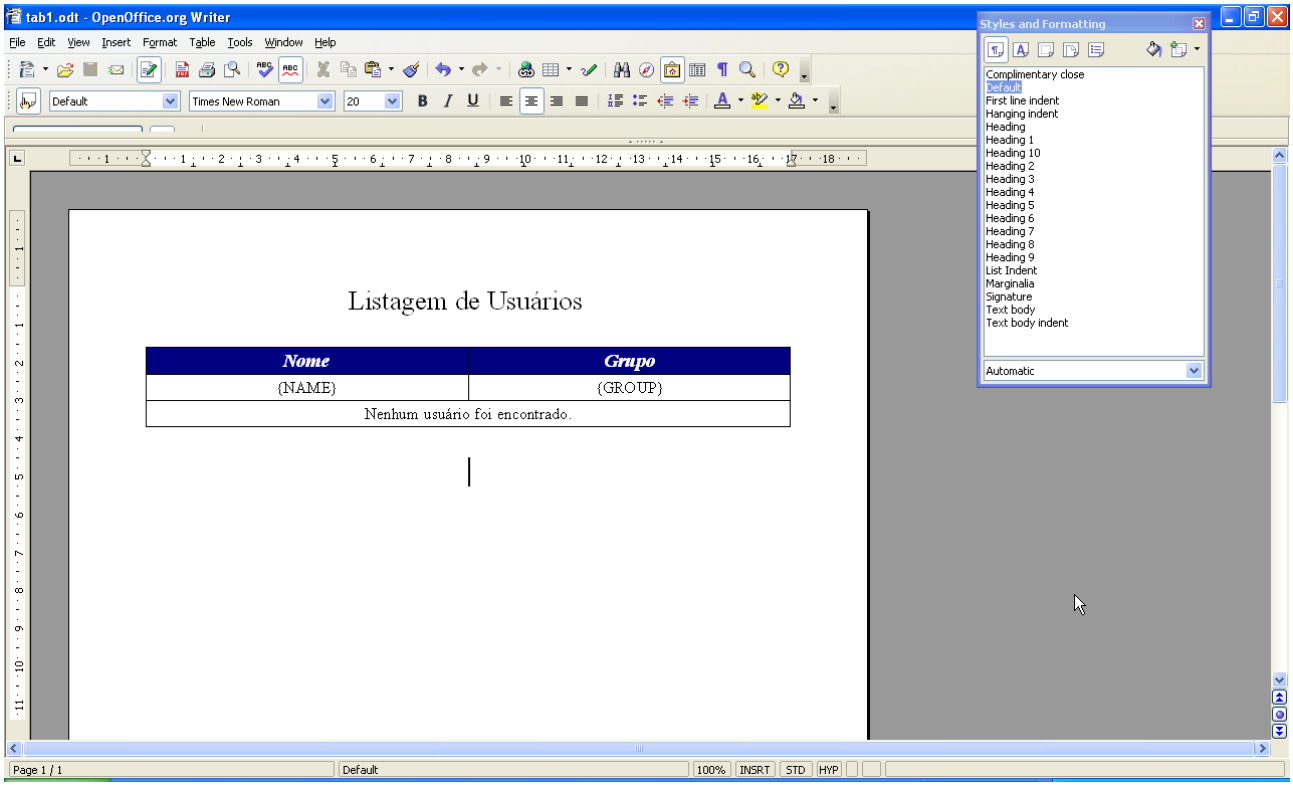
Aqui vale um parênteses. Veremos que não há grandes necessidades de programação para começar a utilizar o método. Graças à filosofia *Open Source*, o trabalho pesado já foi feito por várias pessoas. Agora basta procurar e juntar as peças. Já foi dito que o futuro da programação tenderia mais a conectar um diagrama de blocos do que codificar. Bem, nessa caso é exatamente isso.

As Peças

Precisamos de um aplicativo que gere e leia arquivos no formato OpenDocument (OpenOffice), um editor de XML ou de texto, um programa para descompactar arquivos em formato zip. Para o PHP precisamos de um compactador zip e uma ferramenta de *templates*.
Claro, precisamos de um servidor *web* (Apache, por exemplo) com PHP habilitado.
As partes menos comuns podem ser encontradas aqui.

Mãos à obra

Abra o OpenOffice e crie um documento com a tabela, como a seguir, salvando-o com o nome tab1.odt.



Como pode ser percebido, já estamos preparando o arquivo para o uso de templates, marcados pelas chaves {}. Crie uma pasta OpenDocumentFormat em algum lugar do seu *htdocs* e descompacte o arquivo, criando uma nova pasta tab1 (costumo fazer uma cópia do arquivo, mudando a extensão de odt para zip).

Há algumas rotinas para zipar um arquivo, a minha eleita foi a ziplib.php do phpwiki versão 1.3.11.p1 São necessárias 3 mudanças muito simples no arquivo.

Na primeira linha, comente a função rcs_id. Em seguida, na linha 262 substitua

```
    function ZipWriter ($comment = "", $zipname = "archive.zip") {
```

por

```
    function ZipWriter ($comment = "", $zipname = "archive.zip", $mime = "application/zip") {
```

e na linha 269 troque

```
        header("Content-Type: application/zip; name=\"$zipname\"");
```

por

```
        header("Content-Type: $mime; name=\"$zipname\"");
```

Isso não muda o funcionamento seu funcionamento e garantirá o uso da extensão correta em nosso caso. Para evitarmos problemas com caminhos de inclusão (*include_path*), colocaremos as bibliotecas em nossa própria pasta.

O exemplo a seguir mostra o funcionamento da biblioteca, já dentro de nossa situação específica.

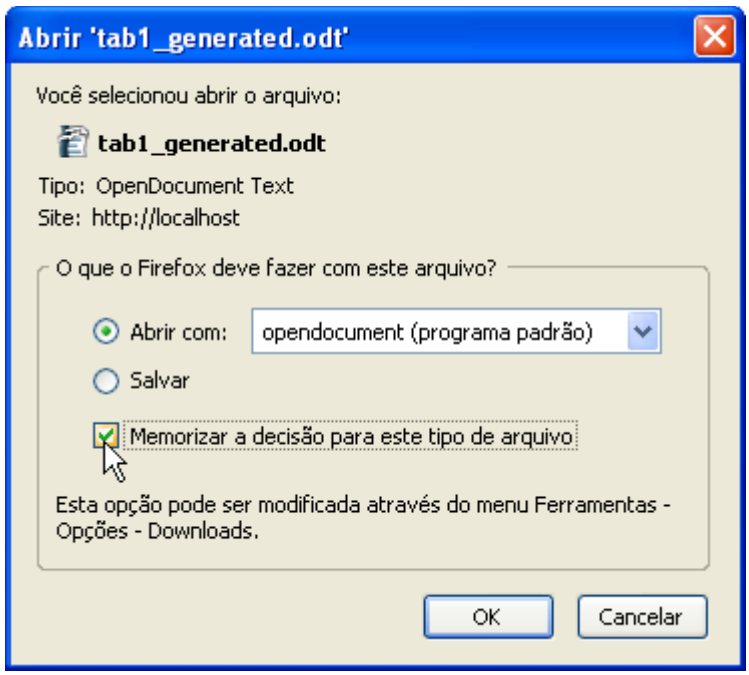
Os arquivos comentados não são necessários e podem gerar erros, além disso não estão presentes em arquivos da versão anterior do OpenOffice.

```
<?php

require_once ("ziplib.php");

$base="tab1";

$zipfile = new ZipWriter("Comentario", $base."_generated.odt",
"application/vnd.oasis.opendocument.text");
$ooofiles = array(
                        "mimetype",
                        "content.xml",
                        "styles.xml",
                        "meta.xml",
                        "settings.xml",
//                      "Configurations2/",
                        "META-INF/manifest.xml",
//                      "Pictures/",
//                      "Thumbnails/thumnail.png"
                             );

foreach ($ooofiles as $file) {
  $handle = fopen("$base/$file","rb");
  $filedata = fread($handle , filesize("$base/$file"));
  $zipfile -> AddRegularFile($file, $filedata);
}

echo $zipfile -> finish();

?>
```

Para fazer o mesmo *script* funcionar com a verão antiga basta mudar o nome do arquivo e sua extensão, bem como o mimetype:

```
$zipfile = new ZipWriter("Comentario", "documento.sxw", "application/vnd.sun.xml.writer");
```

Ao acessar o tab1.php pelo navegador obtemos uma mensagem como a seguinte:



Marque para memorizar a seleção e digite OK. O aquivo deverá ser aberto no OpenOffice, desde que você tenha a versão 2 instalada em sua máquina.

Até esse ponto reolvemos os ítens 1, 2, 4 e 5. Falta o principal, modificar o arquivo desejado.

SmartTemplate

O SmartTemplate é uma ferramenta leve e rápida. Incialmente devemos colocar os arquivos class.smarttemplate.php, class.smarttemplateparser.php e class.smarttemplatedebugger.php no *include_path* ou no diretório atual para pode utilizar a biblioteca. O arquivo tab2.html de template é:

```
<HTML>
<BODY>
<P ALIGN="CENTER"> Listagem de Usu&aacute;rios</P>

<TABLE ALIGN="CENTER" BORDER="1">
  <TR>
    <TH BGCOLOR="#8080FF">
      Nome
    </TH>
    <TH BGCOLOR="#8080FF">
      Grupo
    </TH>
  </TR>

<!-- BEGIN users -->
  <TR>
    <TD>
      {NAME}
    </TD>
    <TD>
      {GROUP}
    </TD>
  </TR>
<!-- END users -->

</TABLE>
</BODY>
</HTML>
```

E o arquivo que o processa é:

```
<?php

require_once "class.smarttemplate.php";

    $x['users'][0]['NAME']='Giovanni';
    $x['users'][0]['GROUP']='Admin';

    $x['users'][1]['NAME']='Leonardo';
    $x['users'][1]['GROUP']='Operações';

    $x['users'][2]['NAME']='Orfeu';
    $x['users'][2]['GROUP']='Desenvolvimento';

    $x['users'][3]['NAME']='Isa';
    $x['users'][3]['GROUP']='Vendas';

$content = new SmartTemplate('tab2.html');
$content->assign($x);
$content->output();

?>
```

Vale notar o que é possível atribuir a uma variável uma tabela inteira, evitando a execução do *loop*: a matriz $x['users'] é substituída em uma linha! Veja, também, que não necessário incluir muitos marcadores no template, o que será importante para a facilidade de criação e manutenção dos documentos-modelo.

A essa altura a idéia está clara. Vamos fazer o mesmo no content.xml. Criamos uma cópia chamada content2.xml, dentro da pasta tab1. Incluímos as duas linhas, conforme a seguir.

```
   (...)
     </table:table-row>
    </table:table-header-rows>
<!-- BEGIN users -->
    <table:table-row>
     <table:table-cell table:style-name="Tabela1.A2" office:value-type="string">
      <text:p text:style-name="P3">{NAME}</text:p>
     </table:table-cell>
     <table:table-cell table:style-name="Tabela1.B2" office:value-type="string">
      <text:p text:style-name="P3">{GROUP}</text:p>
     </table:table-cell>
    </table:table-row>
<!-- END users -->
    <table:table-row>
  (...)
```

Substituímos a primeira linha

```
<?xml version="1.0" encoding="UTF-8"?>

<office:document-content ...
```

por um marcador

```
{XMLHEADER}

<office:document-content ...
```

Salve o arquivo content2.xml. O motivo da mudança é que na primeira linha existem caracteres especiais que geram um erro quando o OpenOffice tenta abrir o arquivo gerado.

O código do tab3.php é

```
<?php

require_once ("ziplib.php");
require_once "class.smarttemplate.php";

$base="tab1";

    $x['users'][0]['NAME']='Giovanni';     $x['users'][0]['GROUP']='Admin';
    $x['users'][1]['NAME']='Leonardo';     $x['users'][1]['GROUP']=utf8_encode('Operações');
    $x['users'][2]['NAME']='Orfeu';        $x['users'][2]['GROUP']='Desenvolvimento';
    $x['users'][3]['NAME']='Isa';          $x['users'][3]['GROUP']='Vendas';

$content = new SmartTemplate("$base/content2.xml");
$content->assign($x);
$content->assign('XMLHEADER', '<?xml version="1.0" encoding="UTF-8"?>');

// $content->output();
// exit;


$zipfile = new ZipWriter("Comentario", $base."_generated.odt",
"application/vnd.oasis.opendocument.text");
$ooofiles = array(
                        "mimetype",
//                      "content.xml", // Note que está comentada!
                        "styles.xml",
                        "meta.xml",
                        "settings.xml",
//                      "Configurations2/",
                        "META-INF/manifest.xml",
//                      "Pictures/",
//                      "Thumbnails/thumnail.png"
                             );


foreach ($ooofiles as $file) {
  $handle = fopen("$base/$file","rb");
  $filedata = fread($handle , filesize("$base/$file"));
  $zipfile -> AddRegularFile($file, $filedata);
}

$zipfile->addRegularFile('content.xml', $content->result() );

echo $zipfile -> finish();

?>
```

Note que, apesar de lermos o arquivo content2.xml, o compactamos com o nome content.xml. Assim, podemos utilizar diversos esqueletos de arquivo, caso estilos e outros detahes inscritos nos outros arquivos não mudem. Não esqueça de comentar o arquivo content.xml do vetor $ooofiles.

Ainda deve ser notado que as linhas que possuem caracteres especiais devem passar por uma codificação UTF8, senão aparecerão erros no arquivo. É claro que o procedimento correto é codificar todos os campos, este é apenas um exemplo didático.

Caso haja algum problema, descomente as linhas a seguir para enviar o resultado para a tela.

```
// $content->output();
// exit;
```

Controle de fluxo

Ainda temos o problema, a tabela pode ser vazia. Criamos um novo tab4.php - essencialmente igual ao tab3.php - que acessa o content3.xml e difere do anterior apenas na seguinte linha.

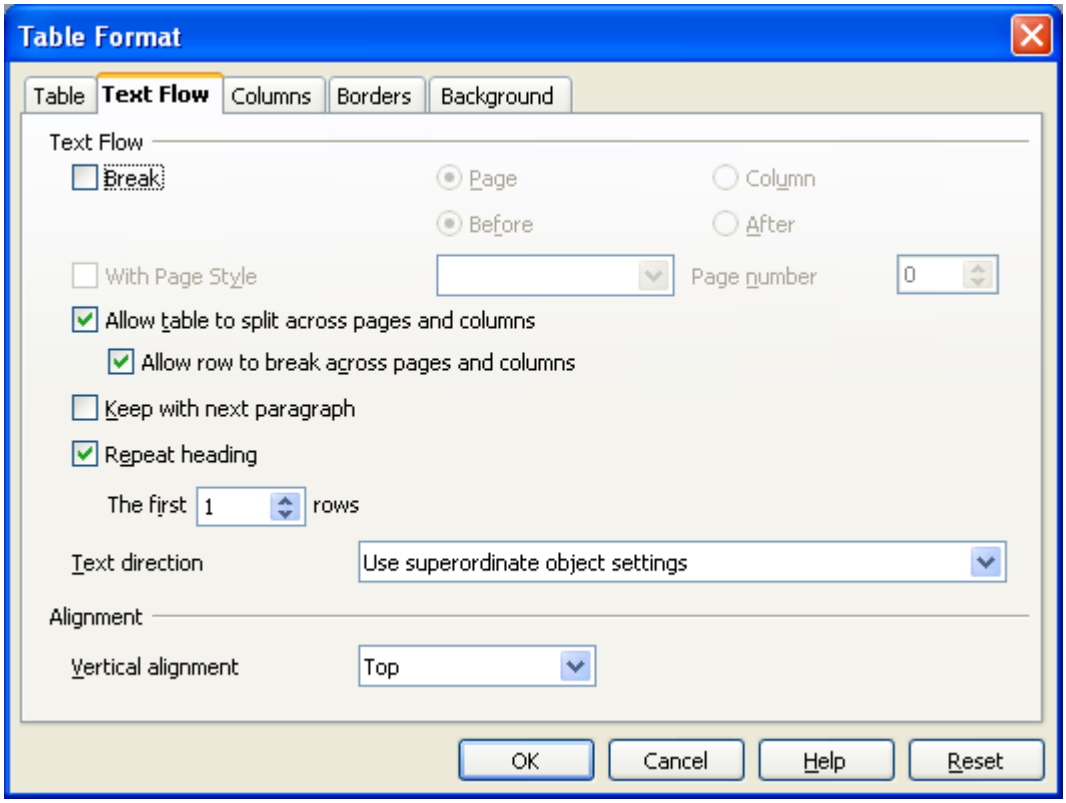
```
    $content = new SmartTemplate("$base/content3.xml");
```

E o respectivo content3.xml possui as seguintes modificações.

```
   (...)
    </table:table-header-rows>
<!-- IF users -->
  <!-- BEGIN users -->
    <table:table-row>
     <table:table-cell table:style-name="Tabela1.A2" office:value-type="string">
      <text:p text:style-name="P3">{NAME}</text:p>
     </table:table-cell>
     <table:table-cell table:style-name="Tabela1.B2" office:value-type="string">
      <text:p text:style-name="P3">{GROUP}</text:p>
     </table:table-cell>
    </table:table-row>
  <!-- END users -->
<!-- ELSE -->
    <table:table-row>
     <table:table-cell table:style-name="Tabela1.B2" table:number-columns-spanned="2" office:value-
type="string">
      <text:p text:style-name="P3">Nenhum usuÃ¡rio foi encontrado.</text:p>
     </table:table-cell>
     <table:covered-table-cell/>
    </table:table-row>
<!-- ENDIF users -->
   </table:table>
   (...)
```

Executando-se o tab4.php, nota-se que a última linha foi removida. Caso se comentem todos os usuários, esvaziando a variável $x a última linha deverá aparecer.

Grandes tabelas podem precisar do dispositivo de repetição de título (*Repeat heading*), disponível no OpenOffice Writer clicando o botão direito sobre a tabela e escolhendo "*Table...*".

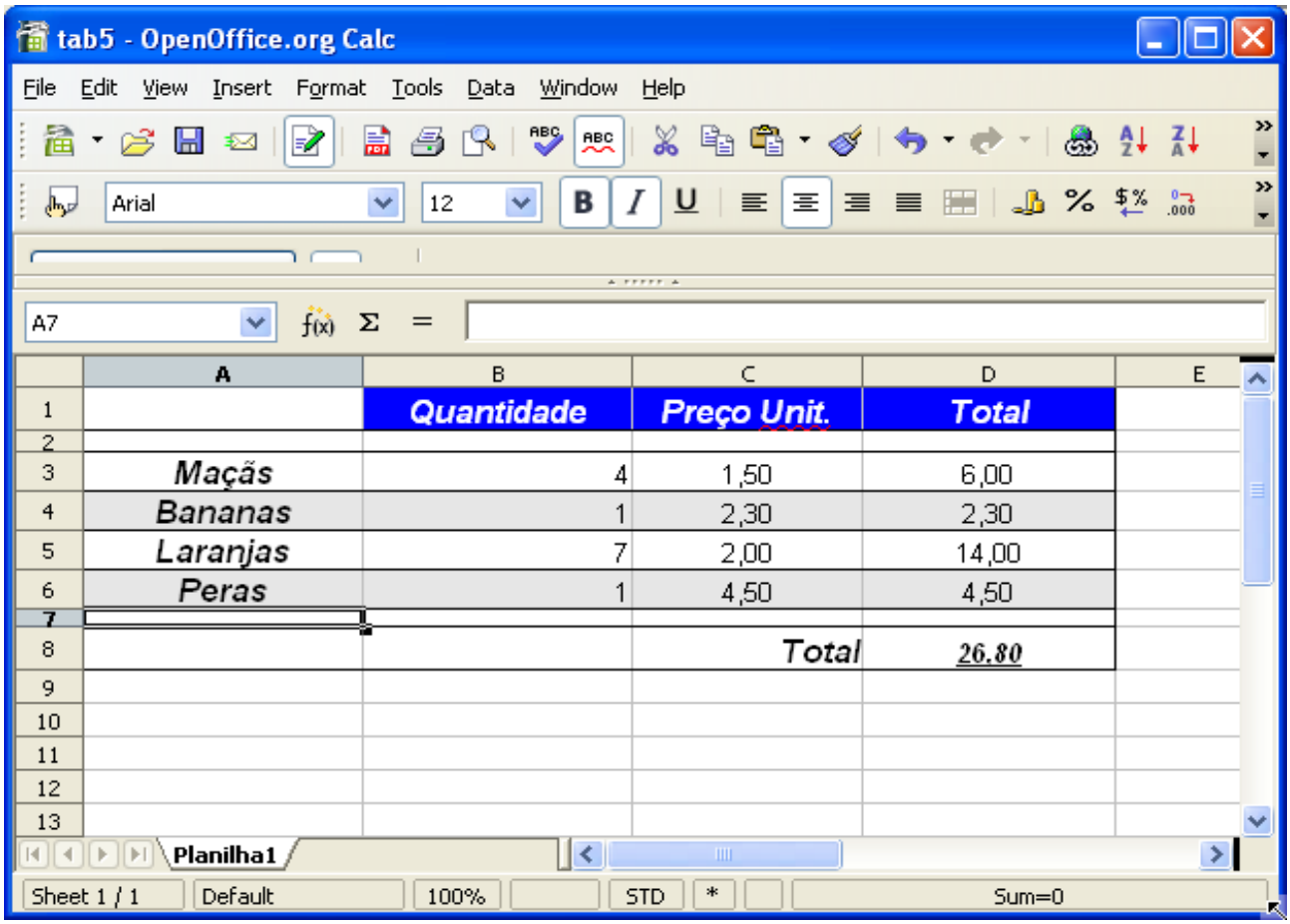


O princípio deve ter sido entendido e pode ser aplicado em qualquer situação. Basta fazer um modelo e copiar os marcadores. Caso sejam necessários estilos, basta criá-los no documento original. Pode-se, também, observar outros arquivos, como o style.xml, comentar a sua entrada no vetor $ooofiles e usar outro *template*.

Gerando Planilhas

Normalmente os relatórios possuem tabelas e texto, mas em alguns momentos é necessário utilizar planilhas, para poder aplicar tabelas pivot ou para gerar gráficos. Pode-se gerar fórmulas, formatações condicionais, ou o que for necessário. O princípio é sempre o mesmo.

Vejamos como isso pode ser feito. Montemos a planilha a seguir.



Observe que a coluna total possui apenas fórmulas e que deixamos uma linha antes e uma linha após os dados. Explicaremos o porquê mais adiante. Vamos usar essa planilha como *template*. Salve-a e descompacte-a em um diretório tab5.

Abrindo o contents.xml, encontramos a seguinte seqüência:

```
(...)
<table:table-row table:style-name="ro3">
  <table:table-cell table:style-name="ce2" office:value-type="string">
   <text:p>MaÃ§Ã£s</text:p>
  </table:table-cell>
  <table:table-cell office:value-type="float" office:value="4">
   <text:p>4</text:p>
  </table:table-cell>
  <table:table-cell office:value-type="float" office:value="1.5">
   <text:p>1,50</text:p>
  </table:table-cell>
  <table:table-cell table:formula="oooc:=[.B3]*[.C3]" office:value-type="float" office:value="6">
   <text:p>6,00</text:p>
  </table:table-cell>
```

```
    </table:table-row>

    <table:table-row table:style-name="ro3">
     <table:table-cell table:style-name="ce3" office:value-type="string">
      <text:p>Bananas</text:p>
     </table:table-cell>
     <table:table-cell table:style-name="ce9" office:value-type="float" office:value="1">
      <text:p>1</text:p>
     </table:table-cell>
     <table:table-cell table:style-name="ce12" office:value-type="float" office:value="2.3">
      <text:p>2,30</text:p>
     </table:table-cell>
     <table:table-cell (...)ormula="oooc:=[.B4]*[.C4]" office:value type="float" office:value="2.3">
      <text:p>2,30</text:p>
     </table:table-cell>
    </table:table-row>
    (...)
```

Note que os valores aparecem duas vezes. Como *office:value-type* e como *text*. O *text* dura apenas até o recálculo, portanto é menos importante. Note que há uma série de estilos – não precisamos de tantos, portanto vamos adotar duas linhas padrão, uma cinza e uma branca – poderíamos ter feito o template menor. Observando os estilos, percebemos que existem vários inúteis, adotaremos apenas dois, o ce2 (branco) e o ce3 (cinza). Esta parte do arquivo pode mudar a depender do que foi feito no OpenOffice Calc.

O bloco central fica:

```
<!-- BEGIN products -->
    <table:table-row table:style-name="ro3">
     <table:table-cell table:style-name="{STYLE}" office:value-type="string">
      <text:p>{NAME}</text:p>
     </table:table-cell>
     <table:table-cell table:style-name="{STYLE}" office:value-type="float" office:value="{QUANT}">
      <text:p>1</text:p>
     </table:table-cell>
     <table:table-cell table:style-name="{STYLE}" office:value-type="float" office:value="{PUNIT}">
      <text:p>4,50</text:p>
     </table:table-cell>
     <table:table-cell table:style-name="{STYLE}" table:formula="oooc:={TOTAL}" office:value-
                                                          type="float" office:value="4.5">
      <text:p>4,50</text:p>
     </table:table-cell>
    </table:table-row>
<!-- END products -->
```

Lembrando de substituir a primeira linha por {XMLHEADER}. A esse ponto o programa tab5.php já pode gerar documentos com número variável de linhas.

```
<?php

require_once ("ziplib.php");
require_once "class.smarttemplate.php";

$base="tab5";

$x['products'][0]['NAME']=utf8_encode('Maças');
$x['products'][0]['QUANT']='2';
$x['products'][0]['PUNIT']='1.5';
$x['products'][0]['TOTAL']='[.B3]*[.C3]';
$x['products'][0]['STYLE']='ce2';

$x['products'][1]['NAME']='Bananas';

    (...)

$x['products'][4]['NAME']='Abacates';
$x['products'][4]['QUANT']='6';
$x['products'][4]['PUNIT']='1.7';
$x['products'][4]['TOTAL']='[.B7]*[.C7]';
$x['products'][4]['STYLE']='ce2';
```

```
$content = new SmartTemplate("$base/content2.xml");
$content->assign($x);
$content->assign('XMLHEADER', '<?xml version="1.0" encoding="UTF-8"?>');

// $content->output();
// exit;


$zipfile = new ZipWriter("Comentario", $base."_generated.odt",
"application/vnd.oasis.opendocument.text");
$ooofiles = array(
                        "mimetype",
//                      "content.xml",
                        "styles.xml",
                        "meta.xml",
                        "settings.xml",
//                      "Configurations2/",
                        "META-INF/manifest.xml",
//                      "Pictures/",
//                      "Thumbnails/thumnail.png"
                );

foreach ($ooofiles as $file) {
  $handle = fopen("$base/$file","rb");
  $filedata = fread($handle , filesize("$base/$file"));
  $zipfile -> AddRegularFile($file, $filedata);
}

$zipfile->addRegularFile('content.xml', $content->result() );

echo $zipfile -> finish();

?>
```

Entretanto, temos 3 coisas a ajustar. A fórmula da última linha deve ser calculada para se adaptar ao número adequado de produtos. A alternânica entre estilos deve ser automática e a fórmula de cada linha também deve ser automatizada. Todos problemas simples.

Os ajustes podem ser vistos no programa tab6.php e seu respectivo content.xml.

```
<?php

require_once ("ziplib.php");
require_once "class.smarttemplate.php";

$base="tab5";

$con = mysql_connect("127.0.0.1", "user", "pass") or die("Não foi possível conectar!");
$base_de_dados = mysql_select_db("open_document",$con) or die ("Base de dados não encontrada!");
$result = mysql_query("SELECT name, quant, punit FROM vendas ORDER BY name") or
          die ("Erro na QUERY!");

# Agora vamos buscar o resultado da query!

$i=0;
while ($line = mysql_fetch_array($result, MYSQL_BOTH)) {
            $x['products'][$i]['NAME']=utf8_encode($line["name"]);
            $x['products'][$i]['QUANT']=utf8_encode($line["quant"]);
            $x['products'][$i]['PUNIT']=utf8_encode($line["punit"]);
            $j=$i+3;
            $x['products'][$i]['TOTAL']="[.B$j]*[.C$j]";
            $x['products'][$i]['STYLE1']= ( $i%2==0 ? 'ce2' : 'ce3' );
            $x['products'][$i]['STYLE2']= ( $i%2==0 ? 'ce8' : 'ce9' );
            $x['products'][$i]['STYLE3']= ( $i%2==0 ? 'ce11' : 'ce12' );
            $i++;
}

$gtotal ="SUM([.D3:.D$j])";

# Não é bom ficar devendo nada ao banco... ;-)
mysql_free_result($result);
mysql_close($con);
```

```
// Gerando o template
$content = new SmartTemplate("$base/content3.xml");
$content->assign('XMLHEADER', '<?xml version="1.0" encoding="UTF-8"?>');
$content->assign($x);
$content->assign('GTOTAL', $gtotal);


// Caso queira debugar.
// $content->output();
// exit;


$zipfile = new ZipWriter("Comentario", $base."_generated.odt",
"application/vnd.oasis.opendocument.text");
$ooofiles = array(
                        "mimetype",
//                      "content.xml",
                        "styles.xml",
                        "meta.xml",
                        "settings.xml",
//                      "Configurations2/",
                        "META-INF/manifest.xml",
//                      "Pictures/",
//                      "Thumbnails/thumnail.png"
                             );

foreach ($ooofiles as $file) {
  $handle = fopen("$base/$file","rb");
  $filedata = fread($handle , filesize("$base/$file"));
  $zipfile -> AddRegularFile($file, $filedata);
}

$zipfile->addRegularFile('content.xml', $content->result() );

echo $zipfile -> finish();

?>
```

E, a parte central do content3.xml (note a mudança nos estilos):

```
 (...)
<!-- BEGIN products -->
    <table:table-row table:style-name="ro3">
     <table:table-cell table:style-name="{STYLE1}" office:value-type="string">
      <text:p>{NAME}</text:p>
     </table:table-cell>
     <table:table-cell table:style-name="{STYLE2}" office:value-type="float" office:value="{QUANT}">
      <text:p>1</text:p>
     </table:table-cell>
     <table:table-cell table:style-name="{STYLE3}" office:value-type="float" office:value="{PUNIT}">
      <text:p>4,50</text:p>
     </table:table-cell>
     <table:table-cell table:style-name="{STYLE3}" table:formula="oooc:={TOTAL}" office:va...
      <text:p>4,50</text:p>
     </table:table-cell>
    </table:table-row>
<!-- END products -->

  (...)

     </table:table-cell>
     <table:table-cell table:style-name="ce20" table:formula="oooc:={GTOTAL}" office:value-
type="float" office:value="26.8">
      <text:p>26,80</text:p>

  (...)
```

Conclusões

Vimos que o formato aberto do OpenOffice e uma porção de peças coletadas torna muito simples a

criação de documentos dinâmicamente gerados, em particular relatórios. A mesma técnica pode ser aplicada para outras linguagens.
Espero que surjam diversos sistemas gerando no formato OpenDocument, ajudando a torná-lo padrão e a difunfir o OpenOffice!

Recursos

[1] http://www.openoffice.org
[2] http://www.oasis-open.org
[3] www.koffice.org
[4] http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf
[5] http://smarty.php.net
[6] cheetah
[7] SmartTemplate: www.smartphp.net ou em http://www.phpclasses.org/browse/package/1032.html.
[8] http://phpwiki.sourceforge.net

[X] www.laslo.org/floridia ou floridia.net – local original deste.

Giovanni Floridia
floridia@gmail.com